```
**********************************************************
    3985 Thu Jun  6 20:37:28 2019
new/usr/src/lib/libm/amd64/src/libm_inlines.h
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

/*
 * Copyright 2011, Richard Lowe.
 */

/* Functions in this file are duplicated in locallibm.il.  Keep them in sync */

#ifndef _LIBM_INLINES_H
#define _LIBM_INLINES_H

#ifdef __GNUC__

#ifdef __cplusplus
extern "C" {
#endif

#include <sys/types.h>
#include <sys/ieeefp.h>

extern __GNU_INLINE float
__inline_sqrtf(float a)
{
        float ret;

        __asm__ __volatile__("sqrtss %1, %0\n\t" : "=x" (ret) : "x" (a));
        return (ret);
}
______unchanged_portion_omitted_
#endif

#endif  /* __GNUC__ */

#endif /* _LIBM_INLINES_H */
```

```
**********************************************************
   22399 Thu Jun  6 20:37:28 2019
new/usr/src/lib/libm/common/C/_SVID_error.c
11210 libm should be cstyle(10NBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

#endif /* ! codereview */
/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2005 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#include "libm.h"
#include "xpg6.h"                       /* __xpg6 */
#include <stdio.h>
#include <float.h>                      /* DBL_MAX, DBL_MIN */
#include <unistd.h>                     /* write */
#if defined(__x86)
#include <ieeefp.h>
#undef  fp_class
#define fp_class        fpclass
#define fp_quiet        FP_QNAN
#endif
#include <errno.h>
#undef fflush
#include <sys/isa_defs.h>


/* INDENT OFF */
/*
 * Report libm exception error according to System V Interface Definition
 * (SVID).
 * Error mapping:
 *      1 -- acos(|x|>1)
 *      2 -- asin(|x|>1)
 *      3 -- atan2(+-0,+-0)
 *      4 -- hypot overflow
 *      5 -- cosh overflow
 *      6 -- exp overflow
 *      7 -- exp underflow
 *      8 -- y0(0)
```



```
 *      9 -- y0(-ve)
 *      10-- y1(0)
 *      11-- y1(-ve)
 *      12-- yn(0)
 *      13-- yn(-ve)
 *      14-- lgamma(finite) overflow
 *      15-- lgamma(-integer)
 *      16-- log(0)
 *      17-- log(x<0)
 *      18-- log10(0)
 *      19-- log10(x<0)
 *      20-- pow(0.0,0.0)
 *      21-- pow(x,y) overflow
 *      22-- pow(x,y) underflow
 *      23-- pow(0,negative)
 *      24-- pow(neg,non-integral)
 *      25-- sinh(finite) overflow
 *      26-- sqrt(negative)
 *      27-- fmod(x,0)
 *      28-- remainder(x,0)
 *      29-- acosh(x<1)
 *      30-- atanh(|x|>1)
 *      31-- atanh(|x|=1)
 *      32-- scalb overflow
 *      33-- scalb underflow
 *      34-- j0(|x|>X_TLOSS)
 *      35-- y0(x>X_TLOSS)
 *      36-- j1(|x|>X_TLOSS)
 *      37-- y1(x>X_TLOSS)
 *      38-- jn(|x|>X_TLOSS, n)
 *      39-- yn(x>X_TLOSS, n)
 *      40-- gamma(finite) overflow
 *      41-- gamma(-integer)
 *      42-- pow(NaN,0.0) return NaN for SVID/XOPEN
 *      43-- log1p(-1)
 *      44-- log1p(x<-1)
 *      45-- logb(0)
 *      46-- nextafter overflow
 *      47-- scalb(x,inf)
 */
/* INDENT ON */

static double setexception(int, double);

static const union {
        unsigned x[2];
        double d;
} C[] = {
______unchanged_portion_omitted_

#define NaN             C[0].d
#define PI_RZ           C[1].d

#define __HI(x)         ((unsigned *)&x)[HIWORD]
#define __LO(x)         ((unsigned *)&x)[LOWORD]
#undef  Inf
#define Inf             HUGE_VAL

double
_SVID_libm_err(double x, double y, int type)
{
_SVID_libm_err(double x, double y, int type) {
        struct exception exc;
        double t, w, ieee_retval = 0;
        enum version lib_version = _lib_version;
        int iy;
```

```
133         /* force libm_ieee behavior in SUSv3 mode */
134         if ((__xpg6 & _C99SUSv3_math_errexcept) != 0)
135                 lib_version = libm_ieee;

137         if (lib_version == c_issue_4)
109         if (lib_version == c_issue_4) {
138                 (void) fflush(stdout);

111         }
140         exc.arg1 = x;
141         exc.arg2 = y;

143 #endif /* ! codereview */
144         switch (type) {
145         case 1:
146                 /* acos(|x|>1) */
147                 exc.type = DOMAIN;
148                 exc.name = "acos";
149                 ieee_retval = setexception(3, 1.0);
150                 exc.retval = 0.0;

152 #endif /* ! codereview */
153                 if (lib_version == strict_ansi) {
154                         errno = EDOM;
155                 } else if (!matherr(&exc)) {
156                         if (lib_version == c_issue_4)
114                         if (lib_version == c_issue_4) {
157                                 (void) write(2, "acos: DOMAIN error\n", 19);

116                         }
159                         errno = EDOM;
160                 }

162 #endif /* ! codereview */
163                 break;
164         case 2:
165                 /* asin(|x|>1) */
166                 exc.type = DOMAIN;
167                 exc.name = "asin";
168                 exc.retval = 0.0;
169                 ieee_retval = setexception(3, 1.0);

171 #endif /* ! codereview */
172                 if (lib_version == strict_ansi) {
173                         errno = EDOM;
174                 } else if (!matherr(&exc)) {
175                         if (lib_version == c_issue_4)
119                         if (lib_version == c_issue_4) {
176                                 (void) write(2, "asin: DOMAIN error\n", 19);

121                         }
178                         errno = EDOM;
179                 }

181 #endif /* ! codereview */
182                 break;
183         case 3:
184                 /* atan2(+-0,+-0) */
185                 exc.arg1 = y;
186                 exc.arg2 = x;
187                 exc.type = DOMAIN;
188                 exc.name = "atan2";
189                 ieee_retval = copysign(1.0, x) == 1.0 ? y : copysign(PI_RZ +
190                     DBL_MIN, y);
124                 ieee_retval = copysign(1.0, x) == 1.0 ? y :
```



```
125                         copysign(PI_RZ + DBL_MIN, y);
191                 exc.retval = 0.0;

193 #endif /* ! codereview */
194                 if (lib_version == strict_ansi) {
195                         errno = EDOM;
196                 } else if (!matherr(&exc)) {
197                         if (lib_version == c_issue_4)
127                         if (lib_version == c_issue_4) {
198                                 (void) write(2, "atan2: DOMAIN error\n", 20);

129                         }
200                         errno = EDOM;
201                 }

203 #endif /* ! codereview */
204                 break;
205         case 4:
206                 /* hypot(finite,finite) overflow */
207                 exc.type = OVERFLOW;
208                 exc.name = "hypot";
209                 ieee_retval = Inf;

211 #endif /* ! codereview */
212                 if (lib_version == c_issue_4)
213                         exc.retval = HUGE;
214                 else
215                         exc.retval = HUGE_VAL;

217 #endif /* ! codereview */
218                 if (lib_version == strict_ansi)
219                         errno = ERANGE;
220                 else if (!matherr(&exc))
221                         errno = ERANGE;

223 #endif /* ! codereview */
224                 break;
225         case 5:
226                 /* cosh(finite) overflow */
227                 exc.type = OVERFLOW;
228                 exc.name = "cosh";
229                 ieee_retval = setexception(2, 1.0);

231 #endif /* ! codereview */
232                 if (lib_version == c_issue_4)
233                         exc.retval = HUGE;
234                 else
235                         exc.retval = HUGE_VAL;

237 #endif /* ! codereview */
238                 if (lib_version == strict_ansi)
239                         errno = ERANGE;
240                 else if (!matherr(&exc))
241                         errno = ERANGE;

243 #endif /* ! codereview */
244                 break;
245         case 6:
246                 /* exp(finite) overflow */
247                 exc.type = OVERFLOW;
248                 exc.name = "exp";
249                 ieee_retval = setexception(2, 1.0);

251 #endif /* ! codereview */
252                 if (lib_version == c_issue_4)
253                         exc.retval = HUGE;
```

```
254                 else
255                         exc.retval = HUGE_VAL;

257 #endif /* ! codereview */
258                 if (lib_version == strict_ansi)
259                         errno = ERANGE;
260                 else if (!matherr(&exc))
261                         errno = ERANGE;

263 #endif /* ! codereview */
264                 break;
265         case 7:
266                 /* exp(finite) underflow */
267                 exc.type = UNDERFLOW;
268                 exc.name = "exp";
269                 ieee_retval = setexception(1, 1.0);
270                 exc.retval = 0.0;

272 #endif /* ! codereview */
273                 if (lib_version == strict_ansi)
274                         errno = ERANGE;
275                 else if (!matherr(&exc))
276                         errno = ERANGE;

278 #endif /* ! codereview */
279                 break;
280         case 8:
281                 /* y0(0) = -inf */
282                 exc.type = DOMAIN;      /* should be SING for IEEE */
283                 exc.name = "y0";
284                 ieee_retval = setexception(0, -1.0);

286 #endif /* ! codereview */
287                 if (lib_version == c_issue_4)
288                         exc.retval = -HUGE;
289                 else
290                         exc.retval = -HUGE_VAL;

292 #endif /* ! codereview */
293                 if (lib_version == strict_ansi) {
294                         errno = EDOM;
295                 } else if (!matherr(&exc)) {
296                         if (lib_version == c_issue_4)
132                         if (lib_version == c_issue_4) {
297                                 (void) write(2, "y0: DOMAIN error\n", 17);

134                         }
299                         errno = EDOM;
300                 }

302 #endif /* ! codereview */
303                 break;
304         case 9:
305                 /* y0(x<0) = NaN */
306                 exc.type = DOMAIN;
307                 exc.name = "y0";
308                 ieee_retval = setexception(3, 1.0);

310 #endif /* ! codereview */
311                 if (lib_version == c_issue_4)
312                         exc.retval = -HUGE;
313                 else
314                         exc.retval = -HUGE_VAL;

316 #endif /* ! codereview */
317                 if (lib_version == strict_ansi) {
```



```
318                         errno = EDOM;
319                 } else if (!matherr(&exc)) {
320                         if (lib_version == c_issue_4)
137                         if (lib_version == c_issue_4) {
321                                 (void) write(2, "y0: DOMAIN error\n", 17);

139                         }
323                         errno = EDOM;
324                 }

326 #endif /* ! codereview */
327                 break;
328         case 10:
329                 /* y1(0) = -inf */
330                 exc.type = DOMAIN;      /* should be SING for IEEE */
331                 exc.name = "y1";
332                 ieee_retval = setexception(0, -1.0);

334 #endif /* ! codereview */
335                 if (lib_version == c_issue_4)
336                         exc.retval = -HUGE;
337                 else
338                         exc.retval = -HUGE_VAL;

340 #endif /* ! codereview */
341                 if (lib_version == strict_ansi) {
342                         errno = EDOM;
343                 } else if (!matherr(&exc)) {
344                         if (lib_version == c_issue_4)
142                         if (lib_version == c_issue_4) {
345                                 (void) write(2, "y1: DOMAIN error\n", 17);

144                         }
347                         errno = EDOM;
348                 }

350 #endif /* ! codereview */
351                 break;
352         case 11:
353                 /* y1(x<0) = NaN */
354                 exc.type = DOMAIN;
355                 exc.name = "y1";
356                 ieee_retval = setexception(3, 1.0);

358 #endif /* ! codereview */
359                 if (lib_version == c_issue_4)
360                         exc.retval = -HUGE;
361                 else
362                         exc.retval = -HUGE_VAL;

364 #endif /* ! codereview */
365                 if (lib_version == strict_ansi) {
366                         errno = EDOM;
367                 } else if (!matherr(&exc)) {
368                         if (lib_version == c_issue_4)
147                         if (lib_version == c_issue_4) {
369                                 (void) write(2, "y1: DOMAIN error\n", 17);

149                         }
371                         errno = EDOM;
372                 }

374 #endif /* ! codereview */
375                 break;
376         case 12:
377                 /* yn(n,0) = -inf */
```

```
378                 exc.type = DOMAIN;      /* should be SING for IEEE */
379                 exc.name = "yn";
380                 ieee_retval = setexception(0, -1.0);

382 #endif /* ! codereview */
383                 if (lib_version == c_issue_4)
384                         exc.retval = -HUGE;
385                 else
386                         exc.retval = -HUGE_VAL;

388 #endif /* ! codereview */
389                 if (lib_version == strict_ansi) {
390                         errno = EDOM;
391                 } else if (!matherr(&exc)) {
392                         if (lib_version == c_issue_4)
152                         if (lib_version == c_issue_4) {
393                                 (void) write(2, "yn: DOMAIN error\n", 17);

154                         }
395                         errno = EDOM;
396                 }

398 #endif /* ! codereview */
399                 break;
400         case 13:
401                 /* yn(x<0) = NaN */
402                 exc.type = DOMAIN;
403                 exc.name = "yn";
404                 ieee_retval = setexception(3, 1.0);

406 #endif /* ! codereview */
407                 if (lib_version == c_issue_4)
408                         exc.retval = -HUGE;
409                 else
410                         exc.retval = -HUGE_VAL;

412 #endif /* ! codereview */
413                 if (lib_version == strict_ansi) {
414                         errno = EDOM;
415                 } else if (!matherr(&exc)) {
416                         if (lib_version == c_issue_4)
157                         if (lib_version == c_issue_4) {
417                                 (void) write(2, "yn: DOMAIN error\n", 17);

159                         }
419                         errno = EDOM;
420                 }

422 #endif /* ! codereview */
423                 break;
424         case 14:
425                 /* lgamma(finite) overflow */
426                 exc.type = OVERFLOW;
427                 exc.name = "lgamma";
428                 ieee_retval = setexception(2, 1.0);

430 #endif /* ! codereview */
431                 if (lib_version == c_issue_4)
432                         exc.retval = HUGE;
433                 else
434                         exc.retval = HUGE_VAL;

436 #endif /* ! codereview */
437                 if (lib_version == strict_ansi)
438                         errno = ERANGE;
439                 else if (!matherr(&exc))
```



```
440                         errno = ERANGE;

442 #endif /* ! codereview */
443                 break;
444         case 15:
445                 /* lgamma(-integer) or lgamma(0) */
446                 exc.type = SING;
447                 exc.name = "lgamma";
448                 ieee_retval = setexception(0, 1.0);

450 #endif /* ! codereview */
451                 if (lib_version == c_issue_4)
452                         exc.retval = HUGE;
453                 else
454                         exc.retval = HUGE_VAL;

456 #endif /* ! codereview */
457                 if (lib_version == strict_ansi) {
458                         errno = EDOM;
459                 } else if (!matherr(&exc)) {
460                         if (lib_version == c_issue_4)
162                         if (lib_version == c_issue_4) {
461                                 (void) write(2, "lgamma: SING error\n", 19);

164                         }
463                         errno = EDOM;
464                 }

466 #endif /* ! codereview */
467                 break;
468         case 16:
469                 /* log(0) */
470                 exc.type = SING;
471                 exc.name = "log";
472                 ieee_retval = setexception(0, -1.0);

474 #endif /* ! codereview */
475                 if (lib_version == c_issue_4)
476                         exc.retval = -HUGE;
477                 else
478                         exc.retval = -HUGE_VAL;

480 #endif /* ! codereview */
481                 if (lib_version == strict_ansi) {
482                         errno = ERANGE;
483                 } else if (!matherr(&exc)) {
484                         if (lib_version == c_issue_4) {
485                                 (void) write(2, "log: SING error\n", 16);
486                                 errno = EDOM;
487                         } else {
488                                 errno = ERANGE;
489                         }
490                 }

492 #endif /* ! codereview */
493                 break;
494         case 17:
495                 /* log(x<0) */
496                 exc.type = DOMAIN;
497                 exc.name = "log";
498                 ieee_retval = setexception(3, 1.0);

500 #endif /* ! codereview */
501                 if (lib_version == c_issue_4)
502                         exc.retval = -HUGE;
503                 else
```

```
504                         exc.retval = -HUGE_VAL;

506 #endif /* ! codereview */
507                 if (lib_version == strict_ansi) {
508                         errno = EDOM;
509                 } else if (!matherr(&exc)) {
510                         if (lib_version == c_issue_4)
167                         if (lib_version == c_issue_4) {
511                                 (void) write(2, "log: DOMAIN error\n", 18);

169                         }
513                         errno = EDOM;
514                 }

516 #endif /* ! codereview */
517                 break;
518         case 18:
519                 /* log10(0) */
520                 exc.type = SING;
521                 exc.name = "log10";
522                 ieee_retval = setexception(0, -1.0);

524 #endif /* ! codereview */
525                 if (lib_version == c_issue_4)
526                         exc.retval = -HUGE;
527                 else
528                         exc.retval = -HUGE_VAL;

530 #endif /* ! codereview */
531                 if (lib_version == strict_ansi) {
532                         errno = ERANGE;
533                 } else if (!matherr(&exc)) {
534                         if (lib_version == c_issue_4) {
535                                 (void) write(2, "log10: SING error\n", 18);
536                                 errno = EDOM;
537                         } else {
538                                 errno = ERANGE;
539                         }
540                 }

542 #endif /* ! codereview */
543                 break;
544         case 19:
545                 /* log10(x<0) */
546                 exc.type = DOMAIN;
547                 exc.name = "log10";
548                 ieee_retval = setexception(3, 1.0);

550 #endif /* ! codereview */
551                 if (lib_version == c_issue_4)
552                         exc.retval = -HUGE;
553                 else
554                         exc.retval = -HUGE_VAL;

556 #endif /* ! codereview */
557                 if (lib_version == strict_ansi) {
558                         errno = EDOM;
559                 } else if (!matherr(&exc)) {
560                         if (lib_version == c_issue_4)
172                         if (lib_version == c_issue_4) {
561                                 (void) write(2, "log10: DOMAIN error\n", 20);

174                         }
563                         errno = EDOM;
564                 }
```



```
566 #endif /* ! codereview */
567                 break;
568         case 20:

570                 /*
571                  * pow(0.0,0.0)
572                  * error only if lib_version == c_issue_4
573                  */
177                 /* pow(0.0,0.0) */
178                 /* error only if lib_version == c_issue_4 */
574                 exc.type = DOMAIN;
575                 exc.name = "pow";
576                 exc.retval = 0.0;
577                 ieee_retval = 1.0;

579 #endif /* ! codereview */
580                 if (lib_version != c_issue_4) {
581                         exc.retval = 1.0;
582                 } else if (!matherr(&exc)) {
583                         (void) write(2, "pow(0,0): DOMAIN error\n", 23);
584                         errno = EDOM;
585                 }

587 #endif /* ! codereview */
588                 break;
589         case 21:
590                 /* pow(x,y) overflow */
591                 exc.type = OVERFLOW;
592                 exc.name = "pow";
593                 exc.retval = (lib_version == c_issue_4) ? HUGE : HUGE_VAL;

183                 exc.retval = (lib_version == c_issue_4)? HUGE : HUGE_VAL;
595                 if (signbit(x)) {
596                         t = rint(y);

598 #endif /* ! codereview */
599                         if (t == y) {
600                                 w = rint(0.5 * y);

602                                 if (t != w + w) /* y is odd */
186                                 if (t != w + w) {       /* y is odd */
603                                         exc.retval = -exc.retval;
604                         }
605                 }

190                 }
607                 ieee_retval = setexception(2, exc.retval);

609 #endif /* ! codereview */
610                 if (lib_version == strict_ansi)
611                         errno = ERANGE;
612                 else if (!matherr(&exc))
613                         errno = ERANGE;

615 #endif /* ! codereview */
616                 break;
617         case 22:
618                 /* pow(x,y) underflow */
619                 exc.type = UNDERFLOW;
620                 exc.name = "pow";
621                 exc.retval = 0.0;

623 #endif /* ! codereview */
624                 if (signbit(x)) {
625                         t = rint(y);
```

```
#endif /* ! codereview */
                        if (t == y) {
                                w = rint(0.5 * y);

#endif /* ! codereview */
                                if (t != w + w) /* y is odd */
                                        exc.retval = -exc.retval;
                        }
                }

#endif /* ! codereview */
                ieee_retval = setexception(1, exc.retval);

#endif /* ! codereview */
                if (lib_version == strict_ansi)
                        errno = ERANGE;
                else if (!matherr(&exc))
                        errno = ERANGE;

#endif /* ! codereview */
                break;
        case 23:
                /* (+-0)**neg */
                exc.type = DOMAIN;
                exc.name = "pow";
                ieee_retval = setexception(0, 1.0);
                {
                        int ahy, k, j, yisint, ly, hx;

                        /* BEGIN CSTYLED */
                        /* INDENT OFF */
                        /*
                         * determine if y is an odd int when x = -0
                         * yisint = 0       ... y is not an integer
                         * yisint = 1       ... y is an odd int
                         * yisint = 2       ... y is an even int
                         */
                        /* END CSTYLED */
                        /* INDENT ON */
                        hx = __HI(x);
                        ahy = __HI(y) & 0x7fffffff;
                        ahy = __HI(y)&0x7fffffff;
                        ly = __LO(y);

                        yisint = 0;

#endif /* ! codereview */
                        if (ahy >= 0x43400000) {
                                yisint = 2; /* even integer y */
                        } else if (ahy >= 0x3ff00000) {
                                k = (ahy >> 20) - 0x3ff;        /* exponent */

#endif /* ! codereview */
                                if (k > 20) {
                                        j = ly >> (52 - k);

#endif /* ! codereview */
                                        if ((j << (52 - k)) == ly)
                                                yisint = 2 - (j & 1);
                                } else if (ly == 0) {
                                        j = ahy >> (20 - k);

#endif /* ! codereview */
                                        if ((j << (20 - k)) == ahy)
                                                yisint = 2 - (j & 1);
                                }
```



```
                        }

#endif /* ! codereview */
                        if (hx < 0 && yisint == 1)
                                ieee_retval = -ieee_retval;
                }

#endif /* ! codereview */
                if (lib_version == c_issue_4)
                        exc.retval = 0.0;
                else
                        exc.retval = -HUGE_VAL;

#endif /* ! codereview */
                if (lib_version == strict_ansi) {
                        errno = EDOM;
                } else if (!matherr(&exc)) {
                        if (lib_version == c_issue_4) {
                                (void) write(2, "pow(0,neg): DOMAIN error\n",
                                    25);
                        }

#endif /* ! codereview */
                        errno = EDOM;
                }

#endif /* ! codereview */
                break;
        case 24:
                /* neg**non-integral */
                exc.type = DOMAIN;
                exc.name = "pow";
                ieee_retval = setexception(3, 1.0);

#endif /* ! codereview */
                if (lib_version == c_issue_4)
                        exc.retval = 0.0;
                else
                        exc.retval = ieee_retval;       /* X/Open allow NaN */

#endif /* ! codereview */
                if (lib_version == strict_ansi) {
                        errno = EDOM;
                } else if (!matherr(&exc)) {
                        if (lib_version == c_issue_4) {
                                (void) write(2,
                                    "neg**non-integral: DOMAIN error\n", 32);
                        }

#endif /* ! codereview */
                        errno = EDOM;
                }

#endif /* ! codereview */
                break;
        case 25:
                /* sinh(finite) overflow */
                exc.type = OVERFLOW;
                exc.name = "sinh";
                ieee_retval = copysign(Inf, x);

#endif /* ! codereview */
                if (lib_version == c_issue_4)
                        exc.retval = x > 0.0 ? HUGE : -HUGE;
                else
                        exc.retval = x > 0.0 ? HUGE_VAL : -HUGE_VAL;
```

```
757 #endif /* ! codereview */
758                 if (lib_version == strict_ansi)
759                         errno = ERANGE;
760                 else if (!matherr(&exc))
761                         errno = ERANGE;

763 #endif /* ! codereview */
764                 break;
765         case 26:
766                 /* sqrt(x<0) */
767                 exc.type = DOMAIN;
768                 exc.name = "sqrt";
769                 ieee_retval = setexception(3, 1.0);

771 #endif /* ! codereview */
772                 if (lib_version == c_issue_4)
773                         exc.retval = 0.0;
774                 else
775                         exc.retval = ieee_retval;       /* quiet NaN */

777 #endif /* ! codereview */
778                 if (lib_version == strict_ansi) {
779                         errno = EDOM;
780                 } else if (!matherr(&exc)) {
781                         if (lib_version == c_issue_4)
205                         if (lib_version == c_issue_4) {
782                                 (void) write(2, "sqrt: DOMAIN error\n", 19);

207                         }
784                         errno = EDOM;
785                 }

787 #endif /* ! codereview */
788                 break;
789         case 27:
790                 /* fmod(x,0) */
791                 exc.type = DOMAIN;
792                 exc.name = "fmod";

794 #endif /* ! codereview */
795                 if (fp_class(x) == fp_quiet)
796                         ieee_retval = NaN;
797                 else
798                         ieee_retval = setexception(3, 1.0);

800 #endif /* ! codereview */
801                 if (lib_version == c_issue_4)
802                         exc.retval = x;
803                 else
804                         exc.retval = ieee_retval;

806 #endif /* ! codereview */
807                 if (lib_version == strict_ansi) {
808                         errno = EDOM;
809                 } else if (!matherr(&exc)) {
810                         if (lib_version == c_issue_4)
210                         if (lib_version == c_issue_4) {
811                                 (void) write(2, "fmod:  DOMAIN error\n", 20);

212                         }
813                         errno = EDOM;
814                 }

816 #endif /* ! codereview */
817                 break;
```



```
818         case 28:
819                 /* remainder(x,0) */
820                 exc.type = DOMAIN;
821                 exc.name = "remainder";

823 #endif /* ! codereview */
824                 if (fp_class(x) == fp_quiet)
825                         ieee_retval = NaN;
826                 else
827                         ieee_retval = setexception(3, 1.0);

829 #endif /* ! codereview */
830                 exc.retval = NaN;

832 #endif /* ! codereview */
833                 if (lib_version == strict_ansi) {
834                         errno = EDOM;
835                 } else if (!matherr(&exc)) {
836                         if (lib_version == c_issue_4)
215                         if (lib_version == c_issue_4) {
837                                 (void) write(2, "remainder: DOMAIN error\n",
838                                     24);

218                         }
840                         errno = EDOM;
841                 }

843 #endif /* ! codereview */
844                 break;
845         case 29:
846                 /* acosh(x<1) */
847                 exc.type = DOMAIN;
848                 exc.name = "acosh";
849                 ieee_retval = setexception(3, 1.0);
850                 exc.retval = NaN;

852 #endif /* ! codereview */
853                 if (lib_version == strict_ansi) {
854                         errno = EDOM;
855                 } else if (!matherr(&exc)) {
856                         if (lib_version == c_issue_4)
221                         if (lib_version == c_issue_4) {
857                                 (void) write(2, "acosh: DOMAIN error\n", 20);

223                         }
859                         errno = EDOM;
860                 }

862 #endif /* ! codereview */
863                 break;
864         case 30:
865                 /* atanh(|x|>1) */
866                 exc.type = DOMAIN;
867                 exc.name = "atanh";
868                 ieee_retval = setexception(3, 1.0);
869                 exc.retval = NaN;

871 #endif /* ! codereview */
872                 if (lib_version == strict_ansi) {
873                         errno = EDOM;
874                 } else if (!matherr(&exc)) {
875                         if (lib_version == c_issue_4)
226                         if (lib_version == c_issue_4) {
876                                 (void) write(2, "atanh: DOMAIN error\n", 20);

228                         }
```

```
                        errno = EDOM;
                }

#endif /* ! codereview */
                break;
        case 31:
                /* atanh(|x|=1) */
                exc.type = SING;
                exc.name = "atanh";
                ieee_retval = setexception(0, x);
                exc.retval = ieee_retval;

#endif /* ! codereview */
                if (lib_version == strict_ansi) {
                        errno = ERANGE;
                } else if (!matherr(&exc)) {
                        if (lib_version == c_issue_4) {
                                (void) write(2, "atanh: SING error\n", 18);
                                errno = EDOM;
                        } else {
                                errno = ERANGE;
                        }
                }

#endif /* ! codereview */
                break;
        case 32:
                /* scalb overflow; SVID also returns +-HUGE_VAL */
                exc.type = OVERFLOW;
                exc.name = "scalb";
                ieee_retval = setexception(2, x);
                exc.retval = x > 0.0 ? HUGE_VAL : -HUGE_VAL;

#endif /* ! codereview */
                if (lib_version == strict_ansi)
                        errno = ERANGE;
                else if (!matherr(&exc))
                        errno = ERANGE;

#endif /* ! codereview */
                break;
        case 33:
                /* scalb underflow */
                exc.type = UNDERFLOW;
                exc.name = "scalb";
                ieee_retval = setexception(1, x);
                exc.retval = ieee_retval;       /* +-0.0 */

#endif /* ! codereview */
                if (lib_version == strict_ansi)
                        errno = ERANGE;
                else if (!matherr(&exc))
                        errno = ERANGE;

#endif /* ! codereview */
                break;
        case 34:
                /* j0(|x|>X_TLOSS) */
                exc.type = TLOSS;
                exc.name = "j0";
                exc.retval = 0.0;
                ieee_retval = y;

#endif /* ! codereview */
                if (lib_version == strict_ansi) {
                        errno = ERANGE;
```



```
                } else if (!matherr(&exc)) {
                        if (lib_version == c_issue_4) {
                                (void) write(2, exc.name, 2);
                                (void) write(2, ": TLOSS error\n", 14);
                        }

#endif /* ! codereview */
                        errno = ERANGE;
                }

#endif /* ! codereview */
                break;
        case 35:
                /* y0(x>X_TLOSS) */
                exc.type = TLOSS;
                exc.name = "y0";
                exc.retval = 0.0;
                ieee_retval = y;

#endif /* ! codereview */
                if (lib_version == strict_ansi) {
                        errno = ERANGE;
                } else if (!matherr(&exc)) {
                        if (lib_version == c_issue_4) {
                                (void) write(2, exc.name, 2);
                                (void) write(2, ": TLOSS error\n", 14);
                        }

#endif /* ! codereview */
                        errno = ERANGE;
                }

#endif /* ! codereview */
                break;
        case 36:
                /* j1(|x|>X_TLOSS) */
                exc.type = TLOSS;
                exc.name = "j1";
                exc.retval = 0.0;
                ieee_retval = y;

#endif /* ! codereview */
                if (lib_version == strict_ansi) {
                        errno = ERANGE;
                } else if (!matherr(&exc)) {
                        if (lib_version == c_issue_4) {
                                (void) write(2, exc.name, 2);
                                (void) write(2, ": TLOSS error\n", 14);
                        }

#endif /* ! codereview */
                        errno = ERANGE;
                }

#endif /* ! codereview */
                break;
        case 37:
                /* y1(x>X_TLOSS) */
                exc.type = TLOSS;
                exc.name = "y1";
                exc.retval = 0.0;
                ieee_retval = y;

#endif /* ! codereview */
                if (lib_version == strict_ansi) {
                        errno = ERANGE;
```

```
1010                 } else if (!matherr(&exc)) {
1011                         if (lib_version == c_issue_4) {
1012                                 (void) write(2, exc.name, 2);
1013                                 (void) write(2, ": TLOSS error\n", 14);
1014                         }

1016 #endif /* ! codereview */
1017                         errno = ERANGE;
1018                 }

1020 #endif /* ! codereview */
1021                 break;
1022         case 38:

1024                 /*
1025                  * jn(|x|>X_TLOSS)
1026                  * incorrect ieee value: ieee should never be here
1027                  */
 231                 /* jn(|x|>X_TLOSS) */
 232                 /* incorrect ieee value: ieee should never be here */
1028                 exc.type = TLOSS;
1029                 exc.name = "jn";
1030                 exc.retval = 0.0;
1031                 ieee_retval = 0.0;      /* shall not be used */

1033 #endif /* ! codereview */
1034                 if (lib_version == strict_ansi) {
1035                         errno = ERANGE;
1036                 } else if (!matherr(&exc)) {
1037                         if (lib_version == c_issue_4) {
1038                                 (void) write(2, exc.name, 2);
1039                                 (void) write(2, ": TLOSS error\n", 14);
1040                         }

1042 #endif /* ! codereview */
1043                         errno = ERANGE;
1044                 }

1046 #endif /* ! codereview */
1047                 break;
1048         case 39:

1050                 /*
1051                  * yn(x>X_TLOSS)
1052                  * incorrect ieee value: ieee should never be here
1053                  */
 237                 /* yn(x>X_TLOSS) */
 238                 /* incorrect ieee value: ieee should never be here */
1054                 exc.type = TLOSS;
1055                 exc.name = "yn";
1056                 exc.retval = 0.0;
1057                 ieee_retval = 0.0;      /* shall not be used */

1059 #endif /* ! codereview */
1060                 if (lib_version == strict_ansi) {
1061                         errno = ERANGE;
1062                 } else if (!matherr(&exc)) {
1063                         if (lib_version == c_issue_4) {
1064                                 (void) write(2, exc.name, 2);
1065                                 (void) write(2, ": TLOSS error\n", 14);
1066                         }

1068 #endif /* ! codereview */
1069                         errno = ERANGE;
1070                 }
```



```
1072 #endif /* ! codereview */
1073                 break;
1074         case 40:
1075                 /* gamma(finite) overflow */
1076                 exc.type = OVERFLOW;
1077                 exc.name = "gamma";
1078                 ieee_retval = setexception(2, 1.0);

1080 #endif /* ! codereview */
1081                 if (lib_version == c_issue_4)
1082                         exc.retval = HUGE;
1083                 else
1084                         exc.retval = HUGE_VAL;

1086 #endif /* ! codereview */
1087                 if (lib_version == strict_ansi)
1088                         errno = ERANGE;
1089                 else if (!matherr(&exc))
1090                         errno = ERANGE;

1092 #endif /* ! codereview */
1093                 break;
1094         case 41:
1095                 /* gamma(-integer) or gamma(0) */
1096                 exc.type = SING;
1097                 exc.name = "gamma";
1098                 ieee_retval = setexception(0, 1.0);

1100 #endif /* ! codereview */
1101                 if (lib_version == c_issue_4)
1102                         exc.retval = HUGE;
1103                 else
1104                         exc.retval = HUGE_VAL;

1106 #endif /* ! codereview */
1107                 if (lib_version == strict_ansi) {
1108                         errno = EDOM;
1109                 } else if (!matherr(&exc)) {
1110                         if (lib_version == c_issue_4)
 243                         if (lib_version == c_issue_4) {
1111                                 (void) write(2, "gamma: SING error\n", 18);

 245                         }
1113                         errno = EDOM;
1114                 }

1116 #endif /* ! codereview */
1117                 break;
1118         case 42:

1120                 /*
1121                  * pow(NaN,0.0)
1122                  * error if lib_version == c_issue_4 or ansi_1
1123                  */
 248                 /* pow(NaN,0.0) */
 249                 /* error if lib_version == c_issue_4 or ansi_1 */
1124                 exc.type = DOMAIN;
1125                 exc.name = "pow";
1126                 exc.retval = x;
1127                 ieee_retval = 1.0;

1129 #endif /* ! codereview */
1130                 if (lib_version == strict_ansi) {
1131                         exc.retval = 1.0;
1132                 } else if (!matherr(&exc)) {
1133                         if ((lib_version == c_issue_4) || (lib_version ==
```

```
                                ansi_1))
                        if ((lib_version == c_issue_4) || (lib_version == ansi_1
                                errno = EDOM;
                }

#endif /* ! codereview */
                break;
        case 43:
                /* log1p(-1) */
                exc.type = SING;
                exc.name = "log1p";
                ieee_retval = setexception(0, -1.0);

#endif /* ! codereview */
                if (lib_version == c_issue_4)
                        exc.retval = -HUGE;
                else
                        exc.retval = -HUGE_VAL;

#endif /* ! codereview */
                if (lib_version == strict_ansi) {
                        errno = ERANGE;
                } else if (!matherr(&exc)) {
                        if (lib_version == c_issue_4) {
                                (void) write(2, "log1p: SING error\n", 18);
                                errno = EDOM;
                        } else {
                                errno = ERANGE;
                        }
                }

#endif /* ! codereview */
                break;
        case 44:
                /* log1p(x<-1) */
                exc.type = DOMAIN;
                exc.name = "log1p";
                ieee_retval = setexception(3, 1.0);
                exc.retval = ieee_retval;

#endif /* ! codereview */
                if (lib_version == strict_ansi) {
                        errno = EDOM;
                } else if (!matherr(&exc)) {
                        if (lib_version == c_issue_4)
                        if (lib_version == c_issue_4) {
                                (void) write(2, "log1p: DOMAIN error\n", 20);

                        }
                        errno = EDOM;
                }

#endif /* ! codereview */
                break;
        case 45:
                /* logb(0) */
                exc.type = DOMAIN;
                exc.name = "logb";
                ieee_retval = setexception(0, -1.0);
                exc.retval = -HUGE_VAL;

#endif /* ! codereview */
                if (lib_version == strict_ansi)
                        errno = EDOM;
                else if (!matherr(&exc))
                        errno = EDOM;
```



```
#endif /* ! codereview */
                break;
        case 46:
                /* nextafter overflow */
                exc.type = OVERFLOW;
                exc.name = "nextafter";

#endif /* ! codereview */
                /*
                 * The value as returned by setexception is +/-DBL_MAX in
                 * round-to-{zero,-/+Inf} mode respectively, which is not
                 * usable.
                 */
                (void) setexception(2, x);
                ieee_retval = x > 0 ? Inf : -Inf;
                exc.retval = x > 0 ? HUGE_VAL : -HUGE_VAL;

#endif /* ! codereview */
                if (lib_version == strict_ansi)
                        errno = ERANGE;
                else if (!matherr(&exc))
                        errno = ERANGE;

#endif /* ! codereview */
                break;
        case 47:
                /* scalb(x,inf) */
                iy = ((int *)&y)[HIWORD];

#endif /* ! codereview */
                if (lib_version == c_issue_4)
                        /* SVID3: ERANGE in all cases */
                        errno = ERANGE;
                else if ((x == 0.0 && iy > 0) || (!finite(x) && iy < 0))
                        /* EDOM for scalb(0,+inf) or scalb(inf,-inf) */
                        errno = EDOM;

                exc.retval = ieee_retval = ((iy < 0) ? x / -y : x * y);
                exc.retval = ieee_retval = ((iy < 0)? x / -y : x * y);
                break;
        }

#endif /* ! codereview */
        switch (lib_version) {
        case c_issue_4:
        case ansi_1:
        case strict_ansi:
                return (exc.retval);
        /* NOTREACHED */
        default:
                return (ieee_retval);
        }

#endif /* ! codereview */
        /* NOTREACHED */
}

static double
setexception(int n, double x)
{
setexception(int n, double x) {
        /*
         * n =
         * 0    division by zero
         * 1    underflow
```

```
 * 2    overflow
 * 3    invalid
 */
volatile double one = 1.0, zero = 0.0, retv;

switch (n) {
case 0:                         /* division by zero */
        retv = copysign(one / zero, x);
        break;
case 1:                         /* underflow */
        retv = DBL_MIN * copysign(DBL_MIN, x);
        break;
case 2:                         /* overflow */
        retv = DBL_MAX * copysign(DBL_MAX, x);
        break;
case 3:                         /* invalid */
        retv = zero * Inf;      /* for Cheetah */
        break;
}
```

```
#endif /* ! codereview */
        return (retv);
}
```

```
**********************************************************
    6859 Thu Jun  6 20:37:29 2019
new/usr/src/lib/libm/common/C/_TBL_atan.c
11210 libm should be cstyle(10NBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#include "libm_protos.h"

/*
 * Let y[j] = _TBL_atan[2j], atan_y[j] = _TBL_atan[2j+1], j = 0, 1, ..., 95.
 * {y[j], 0 <= j < 96} is a set of break points in (-1/8, 8) chosen so that
 * the high part of y[j] is very close to 0x3fc08000 + (j << 16),
 * and atan_y[j] = atan(y[j]) rounded has relative error bounded by 2^-60.
 *
 * -- K.C. Ng, 10/17/2004
 */

const double _TBL_atan[] = {
        1.28906287871928065814e-01, 1.28199318484201185697e-01,
        1.36718905591866640714e-01, 1.35876480966603985223e-01,
        1.44531257606217988787e-01, 1.43537301152401930437e-01,
        1.52343679482641575218e-01, 1.51181262880709432750e-01,
        1.60156177403962790562e-01, 1.58807537535115006477e-01,
        1.67968772982362929413e-01, 1.66415323534856884891e-01,
        1.75781211596017922227e-01, 1.74003563682464612583e-01,
        1.83593807762862160082e-01, 1.81571767039387044207e-01,
        1.91406205589629646591e-01, 1.89118806085245338977e-01,
        1.99218440148815872925e-01, 1.96643947167121080355e-01,
        2.07031180070658488157e-01, 2.04147078126891479144e-01,
        2.14843557086546094181e-01, 2.11626624363759674452e-01,
        2.22656308649619494311e-01, 2.19082566659412503185e-01,
        2.30468759807905931858e-01, 2.26513550670145669130e-01,
        2.38281413377399470255e-01, 2.33919360814280885563e-01,
        2.46093763828156536499e-01, 2.41298839969374956382e-01,
        2.57812599322508773092e-01, 2.52318074018685223336e-01,
        2.73437443946477509726e-01, 2.66912935433335718471e-01,
```



```
        2.89062532292519769328e-01, 2.81392462451501401688e-01,
        3.04687577351389293767e-01, 2.95751756530947318424e-01,
        3.20312405527377053183e-01, 3.09986305565206343715e-01,
        3.35937715576634265968e-01, 3.24092664204967739749e-01,
        3.51562621385942464247e-01, 3.38066230870244233131e-01,
        3.67187719833070636000e-01, 3.51904019130060419229e-01,
        3.82812538440931826589e-01, 3.65602365234580339859e-01,
        3.98437724467857745658e-01, 3.79158862748537828224e-01,
        4.14062683287296784407e-01, 3.92570291474021892952e-01,
        4.29687654458357937148e-01, 4.05834423459965343284e-01,
        4.45312642848883721847e-01, 4.18949086342842669239e-01,
        4.60937644536906665493e-01, 4.31912354681638355203e-01,
        4.76563149131543906112e-01, 4.44722952952162131623e-01,
        4.92187842452541601812e-01, 4.57378374341803173309e-01,
        5.15624825518001039804e-01, 4.76069192487019954285e-01,
        5.46874516057966109095e-01, 5.00440440618262982753e-01,
        5.78125566624434150675e-01, 5.24180053466007933594e-01,
        6.09375102172641347487e-01, 5.47284455493244337276e-01,
        6.40624936950189516338e-01, 5.69756408779493739303e-01,
        6.71875248719545625775e-01, 5.91599881698465779323e-01,
        7.03124988865964306584e-01, 6.12820194714659649549e-01,
        7.34376295967088421612e-01, 6.33426724884753156175e-01,
        7.65624929092156736310e-01, 6.53426296477277901431e-01,
        7.96874196003358736817e-01, 6.72832055855442590087e-01,
        8.28125565205639735389e-01, 6.91656957129326954714e-01,
        8.59375453355927021448e-01, 7.09911879233846576653e-01,
        8.90625694745052709500e-01, 7.27611720056701827275e-01,
        9.21875110259870345075e-01, 7.44770185320721367361e-01,
        9.53125042657123722201e-01, 7.61402792157321428590e-01,
        9.84374765277631902372e-01, 7.77524191164056688308e-01,
        1.03126494373528343473e+00, 8.00788807142382097481e-01,
        1.09374968909110092952e+00, 8.30144253291031475328e-01,
        1.15625019152505204012e+00, 8.57735575892430546219e-01,
        1.21874985186151341132e+00, 8.83672057048812575886e-01,
        1.28124876006842702836e+00, 9.08066349515326720621e-01,
        1.34375006271148444981e+00, 9.31026566320014126177e-01,
        1.40627222899692072566e+00, 9.52659566341466756967e-01,
        1.46874957658300542285e+00, 9.73037801091363618866e-01,
        1.53124999999999555911e+00, 9.92272112377190040888e-01,
        1.59375089676214143353e+00, 1.01043670320979472876e+00,
        1.65624949800269094524e+00, 1.02760661639661776690e+00,
        1.71874946971376685312e+00, 1.04385296549501305208e+00,
        1.78125111924655166185e+00, 1.05924046784549474864e+00,
        1.84374921332370989013e+00, 1.07382754310190620117e+00,
        1.90625055239083862624e+00, 1.08767078118685489585e+00,
        1.96874992734227549640e+00, 1.10081967347672460278e+00,
        2.06250046973591683042e+00, 1.11934332464931074469e+00,
        2.18749905173933534286e+00, 1.14201813543610697366e+00,
        2.31249933788800232648e+00, 1.16264711873167669864e+00,
        2.43749855191054187742e+00, 1.18147939634549814514e+00,
        2.56251104936881235474e+00, 1.19873002825057639598e+00,
        2.68750036758144528193e+00, 1.21457671610223272296e+00,
        2.81249907059852954916e+00, 1.22918073183895870670e+00,
        2.93749583903062294610e+00, 1.24267599964591468620e+00,
        3.06250108260464948273e+00, 1.25518076906426045980e+00,
        3.18750016629930410517e+00, 1.26679540235591403530e+00,
        3.31250071362610132297e+00, 1.27760948984166233799e+00,
        3.43749999999999333866e+00, 1.28770054149540058575e+00,
        3.56249877589327157423e+00, 1.29713691630583838332e+00,
        3.68750696071718842006e+00, 1.30597947372626776996e+00,
        3.81250023149192607264e+00, 1.31427972905173717777e+00,
        3.93749827850909683846e+00, 1.32208623339324304879e+00,
        4.12500187917697846984e+00, 1.33296050364557672196e+00,
        4.37499759905160701123e+00, 1.34608503917096200553e+00,
        4.62500066729278191957e+00, 1.35785800701782477518e+00,
        4.87499852385410648026e+00, 1.36847463881194641999e+00,
```

```
        5.1249991874211007214 5e+00, 1.3780955383301858319 1e+00,
        5.3750000000000452971 0e+00, 1.3868528702577229694 3e+00,
        5.6249999999999182875 9e+00, 1.3948567013423662786 0e+00,
        5.8749941785409669492 4e+00, 1.4021992232726923077 7e+00,
        6.1250000000001323385 8e+00, 1.4089588955564771310 9e+00,
        6.3749999999999182875 9e+00, 1.4152014988178649446 1e+00,
        6.6249993310776158494 9e+00, 1.4209838553208378186 8e+00,
        6.8750043152859374728 8e+00, 1.4263548378272226102 6e+00,
        7.1249922863288386309 9e+00, 1.4313561206919445112 4e+00,
        7.3749925715454720531 7e+00, 1.4360249082067113590 7e+00,
        7.6249991187360741662 4e+00, 1.4403930040046013516 5e+00,
        7.8750000000001891820 0e+00, 1.4444882097316593672 1e+00,
 25 1.2890628787192806581 4e-01,  1.2819931848420118569 7e-01,
 26 1.3671890559186664071 4e-01,  1.3587648096660398522 3e-01,
 27 1.4453125760621798878 7e-01,  1.4353730115240193043 7e-01,
 28 1.5234367948264157521 8e-01,  1.5118126288070943275 0e-01,
 29 1.6015617740396279056 2e-01,  1.5880753753511500647 7e-01,
 30 1.6796877298236292941 3e-01,  1.6641532353485688489 1e-01,
 31 1.7578121159601792222 7e-01,  1.7400356368246461258 3e-01,
 32 1.8359380776286216008 2e-01,  1.8157176703938704420 7e-01,
 33 1.9140620558962964659 1e-01,  1.8911880608524533897 7e-01,
 34 1.9921844014881587292 5e-01,  1.9664394716712108035 5e-01,
 35 2.0703118007065848815 7e-01,  2.0414707812689147914 4e-01,
 36 2.1484355708654609418 1e-01,  2.1162662436375967445 2e-01,
 37 2.2265630864961949431 1e-01,  2.1908256665941250318 5e-01,
 38 2.3046875980790593185 8e-01,  2.2651355067014566913 0e-01,
 39 2.3828141337739947025 5e-01,  2.3391936081428088556 3e-01,
 40 2.4609376382815653649 9e-01,  2.4129883996937495638 2e-01,
 41 2.5781259932250877309 2e-01,  2.5231807401868522333 6e-01,
 42 2.7343744394647750972 6e-01,  2.6691293543333571847 1e-01,
 43 2.8906253229251976932 8e-01,  2.8139246245150140168 8e-01,
 44 3.0468757735138929376 7e-01,  2.9575175653094731842 4e-01,
 45 3.2031240552737705318 3e-01,  3.0998630556520634371 5e-01,
 46 3.3593771557663426596 8e-01,  3.2409266420496773974 9e-01,
 47 3.5156262138594246424 7e-01,  3.3806623087024423313 1e-01,
 48 3.6718771983307063600 0e-01,  3.5190401913006041922 9e-01,
 49 3.8281253844093182658 9e-01,  3.6560236523458033985 9e-01,
 50 3.9843772446785774565 8e-01,  3.7915886274853782822 4e-01,
 51 4.1406268328729678440 7e-01,  3.9257029147402189295 2e-01,
 52 4.2968765445835793714 8e-01,  4.0583442345996534328 4e-01,
 53 4.4531264284888372184 7e-01,  4.1894908634284266923 9e-01,
 54 4.6093764453690666549 3e-01,  4.3191235468163835520 3e-01,
 55 4.7656349131543906112 e-01,  4.4472295295216213162 3e-01,
 56 4.9218784245254160181 2e-01,  4.5737837434180317330 9e-01,
 57 5.1562482551800103980 4e-01,  4.7606919248701995428 5e-01,
 58 5.4687451605796610909 5e-01,  5.0044044061826298275 3e-01,
 59 5.7812556662443415067 5e-01,  5.2418005346600793359 4e-01,
 60 6.0937510217264134748 7e-01,  5.4728445549324433727 6e-01,
 61 6.4062493695018951633 8e-01,  5.6975640877949373930 3e-01,
 62 6.7187524871954562577 5e-01,  5.9159988169846577932 3e-01,
 63 7.0312498886596430658 4e-01,  6.1282019471465964954 9e-01,
 64 7.3437629596708842161 2e-01,  6.3342672488475315617 5e-01,
 65 7.6562492092156736310 e-01,  6.5342629647727790143 1e-01,
 66 7.9687419600335873681 7e-01,  6.7283205585544259008 7e-01,
 67 8.2812556520563973538 9e-01,  6.9165695712932695471 4e-01,
 68 8.5937545335592702144 8e-01,  7.0991187923384657665 3e-01,
 69 8.9062569474505270950 0e-01,  7.2761172005670182727 5e-01,
 70 9.2187511025987034507 5e-01,  7.4477018532072136736 1e-01,
 71 9.5312504265712372220 1e-01,  7.6140279215732142859 0e-01,
 72 9.8437476527763190237 2e-01,  7.7752419116405668830 8e-01,
 73 1.0312649437352834347 3e+00,  8.0078880714238209748 1e-01,
 74 1.0937496890911009295 2e+00,  8.3014425329103147532 8e-01,
 75 1.1562501915250520401 2e+00,  8.5773557589243054621 9e-01,
 76 1.2187498518651134113 2e+00,  8.8367205704881257588 6e-01,
 77 1.2812487600684270283 6e+00,  9.0806634951532672062 1e-01,
 78 1.3437500627114844498 1e+00,  9.3102656632001412617 7e-01,
```



```
 79 1.4062722289969207256 6e+00,  9.5265956634146675696 7e-01,
 80 1.4687495765830054228 5e+00,  9.7303780109136361886 6e-01,
 81 1.5312499999999955591 1e+00,  9.9227211237719004088 8e-01,
 82 1.5937508967621414335 3e+00,  1.0104367032097947287 6e+00,
 83 1.6562494980026909452 4e+00,  1.0276066163966177669 0e+00,
 84 1.7187494697137668531 2e+00,  1.0438529654950130520 8e+00,
 85 1.7812511192465516618 5e+00,  1.0592404678454947486 4e+00,
 86 1.8437492133237098901 3e+00,  1.0738275431019062011 7e+00,
 87 1.9062505523908386262 4e+00,  1.0876707811868548958 5e+00,
 88 1.9687499273422754964 0e+00,  1.1008196734767246027 8e+00,
 89 2.0625004697359168304 2e+00,  1.1193433246493107446 9e+00,
 90 2.1874990517393353428 6e+00,  1.1420181354361069736 6e+00,
 91 2.3124993378880023264 8e+00,  1.1626471187316766986 4e+00,
 92 2.4374985519105418774 2e+00,  1.1814793963454981451 4e+00,
 93 2.5625110493688123547 4e+00,  1.1987300282505763959 8e+00,
 94 2.6875036758144528193 e+00,  1.2145767161022327229 6e+00,
 95 2.8124990705985295491 6e+00,  1.2291807318389587067 0e+00,
 96 2.9374958390306229461 0e+00,  1.2426759996459146862 0e+00,
 97 3.0625010826046494827 3e+00,  1.2551807690642604598 0e+00,
 98 3.1875001662993041051 7e+00,  1.2667954023559140353 0e+00,
 99 3.3125007136261013229 7e+00,  1.2776094898416623379 9e+00,
100 3.4374999999999333866 e+00,  1.2877005414954005857 5e+00,
101 3.5624987758932715742 3e+00,  1.2971369163058383833 2e+00,
102 3.6875069607171884200 6e+00,  1.3059794737262677699 6e+00,
103 3.8125002314919260726 4e+00,  1.3142797290517371777 e+00,
104 3.9374982785090968384 6e+00,  1.3220862333932430487 9e+00,
105 4.1250018791769784698 4e+00,  1.3329605036455767219 6e+00,
106 4.3749975990516070112 3e+00,  1.3460850391709620055 3e+00,
107 4.6250006672927819195 7e+00,  1.3578580070178247751 8e+00,
108 4.8749985238541064802 6e+00,  1.3684746388119464199 9e+00,
109 5.1249991874211007214 5e+00,  1.3780955383301858319 1e+00,
110 5.3750000000000452971 0e+00,  1.3868528702577229694 3e+00,
111 5.6249999999999182875 9e+00,  1.3948567013423662786 0e+00,
112 5.8749941785409669492 4e+00,  1.4021992232726923077 7e+00,
113 6.1250000000001323385 8e+00,  1.4089588955564771310 9e+00,
114 6.3749999999999182875 9e+00,  1.4152014988178649446 1e+00,
115 6.6249993310776158494 9e+00,  1.4209838553208378186 8e+00,
116 6.8750043152859374728 8e+00,  1.4263548378272226102 6e+00,
117 7.1249922863288386309 9e+00,  1.4313561206919445112 4e+00,
118 7.3749925715454720531 7e+00,  1.4360249082067113590 7e+00,
119 7.6249991187360741662 4e+00,  1.4403930040046013516 5e+00,
120 7.8750000000001891820 0e+00,  1.4444882097316593672 1e+00,
140 };
```

_____*unchanged_portion_omitted_*

```
**********************************************************
    4385 Thu Jun  6 20:37:29 2019
new/usr/src/lib/libm/common/C/_TBL_exp2.c
11210 libm should be cstyle(10NBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#include "libm_protos.h"

const double _TBL_exp2_hi[] = {
        1.00000000000000000e+00, 1.01088928605170048e+00,
        1.02189714865411663e+00, 1.03302487902122841e+00,
        1.04427378242741375e+00, 1.05564517836055716e+00,
        1.06714040067682370e+00, 1.07876079775711986e+00,
        1.09050773266525769e+00, 1.10238258330784089e+00,
        1.11438674259589243e+00, 1.12652161860824185e+00,
        1.13878863475669156e+00, 1.15118922995298267e+00,
        1.16372485877757748e+00, 1.17639699165028122e+00,
        1.18920711500272103e+00, 1.20215673145270308e+00,
        1.21524735998046896e+00, 1.22848053610687002e+00,
        1.24185781207348400e+00, 1.25538075702469110e+00,
        1.26905095719173322e+00, 1.28287001607877826e+00,
        1.29683955465100964e+00, 1.31096121152476441e+00,
        1.32523664315974132e+00, 1.33966752405330292e+00,
        1.35425554693689265e+00, 1.36900242297459052e+00,
        1.38390988196383202e+00, 1.39897967253831124e+00,
        1.41421356237309515e+00, 1.42961333839197002e+00,
        1.44518080697704665e+00, 1.46091779418064704e+00,
        1.47682614593949935e+00, 1.49290772829126484e+00,
        1.50916442759342284e+00, 1.52559815074453820e+00,
        1.54221082540794074e+00, 1.55900440023783693e+00,
        1.57598084510788650e+00, 1.59314215134226700e+00,
        1.61049033194925428e+00, 1.62802742185734783e+00,
        1.64575547815396495e+00, 1.66367658032673638e+00,
        1.68179283050742900e+00, 1.70010635371852348e+00,
        1.71861929812247793e+00, 1.73733383527370622e+00,
        1.75625216037329945e+00, 1.77537649252652119e+00,
```



```
        1.79470907500310717e+00, 1.81425217550039886e+00,
        1.83400808640934243e+00, 1.85397912508338547e+00,
        1.87416763411029996e+00, 1.89457598158696561e+00,
        1.91520656139714740e+00, 1.93606179349229435e+00,
        1.95714412417540018e+00, 1.97845602638795093e+00,
 1.00000000000000000e+00, 1.01088928605170048e+00, 1.02189714865411663e+00,
 1.03302487902122841e+00, 1.04427378242741375e+00, 1.05564517836055716e+00,
 1.06714040067682370e+00, 1.07876079775711986e+00, 1.09050773266525769e+00,
 1.10238258330784089e+00, 1.11438674259589243e+00, 1.12652161860824185e+00,
 1.13878863475669156e+00, 1.15118922995298267e+00, 1.16372485877757748e+00,
 1.17639699165028122e+00, 1.18920711500272103e+00, 1.20215673145270308e+00,
 1.21524735998046896e+00, 1.22848053610687002e+00, 1.24185781207348400e+00,
 1.25538075702469110e+00, 1.26905095719173322e+00, 1.28287001607877826e+00,
 1.29683955465100964e+00, 1.31096121152476441e+00, 1.32523664315974132e+00,
 1.33966752405330292e+00, 1.35425554693689265e+00, 1.36900242297459052e+00,
 1.38390988196383202e+00, 1.39897967253831124e+00, 1.41421356237309515e+00,
 1.42961333839197002e+00, 1.44518080697704665e+00, 1.46091779418064704e+00,
 1.47682614593949935e+00, 1.49290772829126484e+00, 1.50916442759342284e+00,
 1.52559815074453820e+00, 1.54221082540794074e+00, 1.55900440023783693e+00,
 1.57598084510788650e+00, 1.59314215134226700e+00, 1.61049033194925428e+00,
 1.62802742185734783e+00, 1.64575547815396495e+00, 1.66367658032673638e+00,
 1.68179283050742900e+00, 1.70010635371852348e+00, 1.71861929812247793e+00,
 1.73733383527370622e+00, 1.75625216037329945e+00, 1.77537649252652119e+00,
 1.79470907500310717e+00, 1.81425217550039886e+00, 1.83400808640934243e+00,
 1.85397912508338547e+00, 1.87416763411029996e+00, 1.89457598158696561e+00,
 1.91520656139714740e+00, 1.93606179349229435e+00, 1.95714412417540018e+00,
 1.97845602638795093e+00,
};

#endif /* ! codereview */
const double _TBL_exp2_lo[] = {
        0.00000000000000000e+00, -1.52347786033685772e-17,
        5.10922502897344389e-17, 7.60083887402708849e-18,
        8.55188970553796366e-17, 1.75932573877209198e-18,
        -7.89985396684158212e-17, -6.65666043605659260e-17,
        -3.04678207981247115e-17, 5.26603687157069439e-17,
        1.04102784568455710e-16, 5.16585675879545612e-17,
        8.91281267602540778e-17, 3.25071021886382721e-17,
        3.82920483692409350e-17, 5.55420325421807896e-17,
        3.98201523146564611e-17, 6.64498149925230124e-17,
        -7.71263069268148813e-17, -1.89878163130252995e-17,
        4.65802759183693679e-17, -6.71138982129687842e-18,
        2.66793213134218610e-18, 1.71359491824356097e-17,
        2.53825027948883150e-17, -7.18153613551945386e-17,
        -2.85873121003886076e-17, 8.92728259483173198e-17,
        7.70094837980298946e-17, 9.59379791911884877e-17,
        -6.77051165879478629e-17, -9.61421320905132307e-17,
        -9.66729331345291345e-17, -1.20316424890536552e-17,
        -3.02375813499398732e-17, -5.60037718607521580e-17,
        -3.48399455689279580e-17, 1.41929201542840358e-17,
        -1.01645532775429504e-16, 1.11795187801605699e-16,
        7.94983480969762086e-17, 3.78120705335752750e-17,
        -1.01369164712783040e-17, -1.00944065423119625e-16,
        2.47071925697978879e-17, -6.71295508470708409e-17,
        -1.01256799136747726e-16, 5.89099269671309967e-17,
        8.19901002058149652e-17, -8.02371937039770025e-18,
        -1.85138041826311099e-17, 3.16438929929295695e-17,
        2.96014069544887331e-17, 6.42973179655657203e-17,
        1.82274584279120868e-17, -9.96953153892034882e-17,
        3.28310722424562659e-17, 9.76188749072759354e-17,
        -6.12276341300414256e-17, 3.40340353521652967e-17,
        -1.06199460561959626e-16, 1.03323859606763257e-16,
        8.96076779103666777e-17, 4.03887531092781666e-17,
 0.00000000000000000e+00,-1.52347786033685772e-17, 5.10922502897344389e-17,
 7.60083887402708849e-18, 8.55188970553796366e-17, 1.75932573877209198e-18,
-7.89985396684158212e-17,-6.65666043605659260e-17,-3.04678207981247115e-17,
```

```
 5.26603687157069439e-17, 1.04102784568455710e-16, 5.16585675879545612e-17,
 8.91281267602540778e-17, 3.25071021886382721e-17, 3.82920483692409350e-17,
 5.55420325421807896e-17, 3.98201523146564611e-17, 6.64498149925230124e-17,
-7.71263069268148813e-17,-1.89878163130252995e-17, 4.65802759183693679e-17,
-6.71138982129687842e-18, 2.66793213134218610e-18, 1.71359491824356097e-17,
 2.53825027948883150e-17,-7.18153613551945386e-17,-2.85873121003886076e-17,
 8.92728259483173198e-17, 7.70094837980298946e-17, 9.59379791911884877e-17,
-6.77051165879478629e-17,-9.61421320905132307e-17,-9.66729331345291345e-17,
-1.20316424890536552e-17,-3.02375813499398732e-17,-5.60037718607521580e-17,
-3.48399455689279580e-17, 1.41929201542840358e-17,-1.01645532775429504e-16,
 1.11795187801605699e-16, 7.94983480969762086e-17, 3.78120705335752750e-17,
-1.01369164712783040e-17,-1.00944065423119625e-16, 2.47071925697978879e-17,
-6.71295508470708409e-17,-1.01256799136747726e-16, 5.89099269671309967e-17,
 8.19901002058149652e-17,-8.02371937039770025e-18,-1.85138041826311099e-17,
 3.16438929929295695e-17, 2.96014069544887331e-17, 6.42973179655657203e-17,
 1.82274584279120868e-17,-9.96953153892034882e-17, 3.28310722424562659e-17,
 9.76188749072759354e-17,-6.12276341300414256e-17, 3.40340353521652967e-17,
-1.06199460561959626e-16, 1.03323859606763257e-16, 8.96076779103666777e-17,
 4.03887531092781666e-17,
};
```

*_____unchanged_portion_omitted_*

```
**********************************************************
    3437 Thu Jun  6 20:37:30 2019
new/usr/src/lib/libm/common/C/_TBL_ipio2.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#include "libm_protos.h"

/*
 * Table of constants for 2/pi, used in __rem_pio2 (trigl) function.
 */

/*
 * 396 Hex digits (476 decimal) of 2/pi
 */
const int _TBL_ipio2_inf[] = {
        0xA2F983, 0x6E4E44, 0x1529FC, 0x2757D1, 0xF534DD, 0xC0DB62,
        0x95993C, 0x439041, 0xFE5163, 0xABDEBB, 0xC561B7, 0x246E3A,
        0x424DD2, 0xE00649, 0x2EEA09, 0xD1921C, 0xFE1DEB, 0x1CB129,
        0xA73EE8, 0x8235F5, 0x2EBB44, 0x84E99C, 0x7026B4, 0x5F7E41,
        0x3991D6, 0x398353, 0x39F49C, 0x845F8B, 0xBDF928, 0x3B1FF8,
        0x97FFDE, 0x05980F, 0xEF2F11, 0x8B5A0A, 0x6D1F6D, 0x367ECF,
        0x27CB09, 0xB74F46, 0x3F669E, 0x5FEA2D, 0x7527BA, 0xC7EBE5,
        0xF17B3D, 0x0739F7, 0x8A5292, 0xEA6BFB, 0x5FB11F, 0x8D5D08,
        0x560330, 0x46FC7B, 0x6BABF0, 0xCFBC20, 0x9AF436, 0x1DA9E3,
        0x91615E, 0xE61B08, 0x659985, 0x5F14A0, 0x68408D, 0xFFD880,
        0x4D7327, 0x310606, 0x1556CA, 0x73A8C9, 0x60E27B, 0xC08C6B,
0xA2F983, 0x6E4E44, 0x1529FC, 0x2757D1, 0xF534DD, 0xC0DB62,
0x95993C, 0x439041, 0xFE5163, 0xABDEBB, 0xC561B7, 0x246E3A,
0x424DD2, 0xE00649, 0x2EEA09, 0xD1921C, 0xFE1DEB, 0x1CB129,
0xA73EE8, 0x8235F5, 0x2EBB44, 0x84E99C, 0x7026B4, 0x5F7E41,
0x3991D6, 0x398353, 0x39F49C, 0x845F8B, 0xBDF928, 0x3B1FF8,
0x97FFDE, 0x05980F, 0xEF2F11, 0x8B5A0A, 0x6D1F6D, 0x367ECF,
0x27CB09, 0xB74F46, 0x3F669E, 0x5FEA2D, 0x7527BA, 0xC7EBE5,
0xF17B3D, 0x0739F7, 0x8A5292, 0xEA6BFB, 0x5FB11F, 0x8D5D08,
0x560330, 0x46FC7B, 0x6BABF0, 0xCFBC20, 0x9AF436, 0x1DA9E3,
```



```
0x91615E, 0xE61B08, 0x659985, 0x5F14A0, 0x68408D, 0xFFD880,
0x4D7327, 0x310606, 0x1556CA, 0x73A8C9, 0x60E27B, 0xC08C6B,
};

#if 0                                   /* remove from SVR4 */
/*
 * 396 Hex digits (476 decimal) of 2/PI, PI = 66 bits of pi
 */
const int _TBL_ipio2_66[] = {
        0xA2F983, 0x6E4E44, 0x152A00, 0x062BC4, 0x0DA276, 0xBED4C1,
        0xFDF905, 0x5CD5BA, 0x767CEC, 0x1F80D6, 0xC26053, 0x3A0070,
        0x107C2A, 0xF68EE9, 0x687B7A, 0xB990AA, 0x38DE4B, 0x96CFF3,
        0x92735E, 0x8B34F6, 0x195BFC, 0x27F88E, 0xA93EC5, 0x3958A5,
        0x3E5D13, 0x1C55A8, 0x5B4A8B, 0xA42E04, 0x12D105, 0x35580D,
        0xF62347, 0x450900, 0xB98BCA, 0xF7E8A4, 0xA2E5D5, 0x69BC52,
        0xF0381D, 0x1A0A88, 0xFE8714, 0x7F6735, 0xBB7D4D, 0xC6F642,
        0xB27E80, 0x6191BF, 0xB6B750, 0x52776E, 0xD60FD0, 0x607DCC,
        0x68BFAF, 0xED69FC, 0x6EB305, 0xD2557D, 0x25BDFB, 0x3E4AA1,
        0x84472D, 0x8B0376, 0xF77740, 0xD290DF, 0x15EC8C, 0x45A5C3,
        0x6181EF, 0xC5E7E8, 0xD8909C, 0xF62144, 0x298428, 0x6E5D9D,
0xA2F983, 0x6E4E44, 0x152A00, 0x062BC4, 0x0DA276, 0xBED4C1,
0xFDF905, 0x5CD5BA, 0x767CEC, 0x1F80D6, 0xC26053, 0x3A0070,
0x107C2A, 0xF68EE9, 0x687B7A, 0xB990AA, 0x38DE4B, 0x96CFF3,
0x92735E, 0x8B34F6, 0x195BFC, 0x27F88E, 0xA93EC5, 0x3958A5,
0x3E5D13, 0x1C55A8, 0x5B4A8B, 0xA42E04, 0x12D105, 0x35580D,
0xF62347, 0x450900, 0xB98BCA, 0xF7E8A4, 0xA2E5D5, 0x69BC52,
0xF0381D, 0x1A0A88, 0xFE8714, 0x7F6735, 0xBB7D4D, 0xC6F642,
0xB27E80, 0x6191BF, 0xB6B750, 0x52776E, 0xD60FD0, 0x607DCC,
0x68BFAF, 0xED69FC, 0x6EB305, 0xD2557D, 0x25BDFB, 0x3E4AA1,
0x84472D, 0x8B0376, 0xF77740, 0xD290DF, 0x15EC8C, 0x45A5C3,
0x6181EF, 0xC5E7E8, 0xD8909C, 0xF62144, 0x298428, 0x6E5D9D,
};

/*
 * 396 Hex digits (476 decimal) of 2/PI, PI = 53 bits of pi
 */
const int _TBL_ipio2_53[] = {
        0xA2F983, 0x6E4E44, 0x16F3C4, 0xEA69B5, 0xD3E131, 0x60E1D2,
        0xD7982A, 0xC031F5, 0xD67BCC, 0xFD1375, 0x60919B, 0x3FA0BB,
        0x612ABB, 0x714F9B, 0x03DA8A, 0xC05948, 0xD023F4, 0x5AFA37,
        0x51631D, 0xCD7A90, 0xC0474A, 0xF6A6F3, 0x1A52E1, 0x5C3927,
        0x3ADA45, 0x4E2DB5, 0x64E8C4, 0x274A5B, 0xB74ADC, 0x1E6591,
        0x2822BE, 0x4771F5, 0x12A63F, 0x83BD35, 0x2488CA, 0x1FE1BE,
        0x42C21A, 0x682569, 0x2AFB91, 0x68ADE1, 0x4A42E5, 0x9BE357,
        0xB79675, 0xCE998A, 0x83AF8B, 0xE645E6, 0xDF0789, 0x9E9747,
        0xAA15FF, 0x358C3F, 0xAF3141, 0x72A3F7, 0x2BF1D4, 0xF3AD96,
        0x7D759F, 0x257FCE, 0x29FB69, 0xB1B42C, 0xC32DE1, 0x8C0BBD,
        0x31EC2F, 0x942026, 0x85DCE7, 0x653FF3, 0x136FA7, 0x0D7A5F,
0xA2F983, 0x6E4E44, 0x16F3C4, 0xEA69B5, 0xD3E131, 0x60E1D2,
0xD7982A, 0xC031F5, 0xD67BCC, 0xFD1375, 0x60919B, 0x3FA0BB,
0x612ABB, 0x714F9B, 0x03DA8A, 0xC05948, 0xD023F4, 0x5AFA37,
0x51631D, 0xCD7A90, 0xC0474A, 0xF6A6F3, 0x1A52E1, 0x5C3927,
0x3ADA45, 0x4E2DB5, 0x64E8C4, 0x274A5B, 0xB74ADC, 0x1E6591,
0x2822BE, 0x4771F5, 0x12A63F, 0x83BD35, 0x2488CA, 0x1FE1BE,
0x42C21A, 0x682569, 0x2AFB91, 0x68ADE1, 0x4A42E5, 0x9BE357,
0xB79675, 0xCE998A, 0x83AF8B, 0xE645E6, 0xDF0789, 0x9E9747,
0xAA15FF, 0x358C3F, 0xAF3141, 0x72A3F7, 0x2BF1D4, 0xF3AD96,
0x7D759F, 0x257FCE, 0x29FB69, 0xB1B42C, 0xC32DE1, 0x8C0BBD,
0x31EC2F, 0x942026, 0x85DCE7, 0x653FF3, 0x136FA7, 0x0D7A5F,
};
```

_____unchanged_portion_omitted_

```
**********************************************************
   21136 Thu Jun  6 20:37:30 2019
new/usr/src/lib/libm/common/C/_TBL_log.c
11210 libm should be cstyle(10NBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#include "libm_protos.h"

/*
 * Table of constants for log, log2, and log10
 * By K.C. Ng, November 21, 2004
 *
 * Y[j], 1/Y[j], log(Y[j]) for j = 0 to 255
 * where HIWORD(Y[j]) ~ 0x3fb8400 + (j<<15)
 * That is, 256 Y[j] space out logrithmically between 0.09375 and 24, and
 * each is chosen so that 1/Y[j] and log(Y[j]) are very close to a IEEE
 * double. In addition, each log(Y[j]) has 3 trailing zeros.
 */
const double _TBL_log[] = {
        9.47265623608246343e-02, 1.05567010464380857e+01,
        -2.35676082856530300e+00, 9.66796869131412717e-02,
        1.03434344062203838e+01, -2.33635196153499791e+00,
        9.86328118117651004e-02, 1.01386139321308306e+01,
        -2.31635129573594156e+00, 1.00585936733578435e-01,
        9.94174764856737347e+00, -2.29674282498938709e+00,
        1.02539062499949152e-01, 9.75238095238578850e+00,
        -2.27751145544242561e+00, 1.04492186859904843e-01,
        9.57009351656812157e+00, -2.25864297726331742e+00,
        1.06445312294918631e-01, 9.39449543094380957e+00,
        -2.24012392529694537e+00, 1.08398437050250693e-01,
        9.22522526350104144e+00, -2.22194160843615762e+00,
        1.10351562442130582e-01, 9.06194690740703912e+00,
        -2.20408398741152212e+00, 1.12304686894746625e-01,
        8.90434787407592943e+00, -2.18653968262558962e+00,
        1.14257811990227776e-01, 8.75213679118525256e+00,
        -2.16929787526329321e+00, 1.16210936696872255e-01,
```



```
        8.60504207627572093e+00, -2.15234831939887172e+00,
        1.18164061975360682e-01, 8.46280995492959498e+00,
        -2.13568126444263484e+00, 1.20117187499996322e-01,
        8.32520325203277523e+00, -2.11928745022706622e+00,
        1.22070312499895098e-01, 8.19200000000703987e+00,
        -2.10315806829801133e+00, 1.24023436774175100e-01,
        8.06299217317146599e+00, -2.08728472499318229e+00,
        1.26953123746900931e-01, 7.87692315467275872e+00,
        -2.06393736501443570e+00, 1.30859374098123454e-01,
        7.64179109744297769e+00, -2.03363201254049386e+00,
        1.34765623780674720e-01, 7.42028992220936967e+00,
        -2.00421812948999545e+00, 1.38671874242985771e-01,
        7.21126764500034501e+00, -1.97564475345722457e+00,
        1.42578124148616536e-01, 7.01369867201821506e+00,
        -1.94786518986246371e+00, 1.46484374166731490e-01,
        6.82666670549979404e+00, -1.92083651719164372e+00,
        1.50390624434435488e-01, 6.64935067435644189e+00,
        -1.89451920694646070e+00, 1.54296874339723084e-01,
        6.48101268596180624e+00, -1.86887677685174936e+00,
        1.58203124999987427e-01, 6.32098765432149001e+00,
        -1.84387547036714849e+00, 1.62109374999815342e-01,
        6.16867469880220742e+00, -1.81948401724404896e+00,
        1.66015624243955634e-01, 6.02352943919619310e+00,
        -1.79567337310324682e+00, 1.69921874302298687e-01,
        5.88505749542848644e+00, -1.77241651049093640e+00,
        1.73828124315277527e-01, 5.75280901142480605e+00,
        -1.74968825924644555e+00, 1.77734374286237506e-01,
        5.62637364896854919e+00, -1.72746512253855222e+00,
        1.81640624146994889e-01, 5.50537636993989743e+00,
        -1.70572513658236602e+00, 1.85546874316304788e-01,
        5.38947370406942916e+00, -1.68444773712372431e+00,
        1.89453124405085355e-01, 5.27835053203882509e+00,
        -1.66361364967629299e+00, 1.93359374570531595e-01,
        5.17171718320401652e+00, -1.64320477712600699e+00,
        1.97265624263334577e-01, 5.06930694962380368e+00,
        -1.62320411193263148e+00, 2.01171874086291030e-01,
        4.97087380898513764e+00, -1.60359564135180399e+00,
        2.05078123979995308e-01, 4.87619050044336610e+00,
        -1.58436427985572159e+00, 2.08984373896073439e-01,
        4.78504675424820736e+00, -1.56549579585994181e+00,
        2.12890623963011144e-01, 4.69724772930228163e+00,
        -1.54697674768135762e+00, 2.16796874723889421e-01,
        4.61261261848719517e+00, -1.52879442500076479e+00,
        2.20703124198150608e-01, 4.53097346778917753e+00,
        -1.51093680996032553e+00, 2.24609374375627030e-01,
        4.45217392541970725e+00, -1.49339249945607477e+00,
        2.28515625000094036e-01, 4.37606837606657528e+00,
        -1.47615069024134016e+00, 2.32421873924349737e-01,
        4.30252102831546246e+00, -1.45920113655598627e+00,
        2.36328123935216378e-01, 4.23140497774241098e+00,
        -1.44253408394829741e+00, 2.40234375000066919e-01,
        4.16260162601510064e+00, -1.42614026966681173e+00,
        2.44140623863132178e-01, 4.09600001907347711e+00,
        -1.41001089239381727e+00, 2.48046874999894917e-01,
        4.03149606299383390e+00, -1.39413753858134015e+00,
        2.53906248590769879e-01, 3.93846156032078243e+00,
        -1.37079018013412401e+00, 2.61718748558906533e-01,
        3.82089554342693294e+00, -1.34048483059486401e+00,
        2.69531249159214337e-01, 3.71014493910979404e+00,
        -1.31107094300173976e+00, 2.77343749428383191e-01,
        3.60563381024826013e+00, -1.28249756949928795e+00,
        2.85156249289339359e-01, 3.50684932380819214e+00,
        -1.25471800582335113e+00, 2.92968749999700462e-01,
        3.41333333333682321e+00, -1.22768933094427446e+00,
        3.00781248554318814e-01, 3.32467534065511261e+00,
        -1.20137202743229921e+00, 3.08593748521894806e-01,
```

```
        3.24050634463533127e+00, -1.17572959680235023e+00,
        3.16406249999639899e-01, 3.16049382716409077e+00,
        -1.15072828980826181e+00, 3.24218749999785061e-01,
        3.08433734939963511e+00, -1.12633683668362750e+00,
        3.32031248841858584e-01, 3.01176471638753718e+00,
        -1.10252619147729547e+00, 3.39843749265406558e-01,
        2.94252874199264314e+00, -1.07926932798654107e+00,
        3.47656249999834799e-01, 2.87640449438338930e+00,
        -1.05654107474789782e+00, 3.55468749999899247e-01,
        2.81318681318761055e+00, -1.03431793796299587e+00,
        3.63281249999864997e-01, 2.75268817204403371e+00,
        -1.01257795132667816e+00, 3.71093749064121570e-01,
        2.69473684890124421e+00, -9.91300555400967731e-01,
        3.78906249999751032e-01, 2.63917525773369288e+00,
        -9.70466465976836723e-01, 3.86718748879039009e-01,
        2.58585859335407608e+00, -9.50075972436191156e-01,
        3.94531249999987899e-01, 2.53465346534661240e+00,
        -9.30056927638333697e-01, 4.02343749999485523e-01,
        2.48543689320706163e+00, -9.10448456251205407e-01,
        4.10156249578856991e-01, 2.43809524059864202e+00,
        -8.91217095348825872e-01, 4.17968749447214571e-01,
        2.39252336765021800e+00, -8.72348611340208357e-01,
        4.25781248601723117e-01, 2.34862386092395203e+00,
        -8.53829565534445223e-01, 4.33593749393073047e-01,
        2.30630630953458038e+00, -8.35647244566987801e-01,
        4.41406248572254134e-01, 2.26548673299152270e+00,
        -8.17789629001761220e-01, 4.49218749348472501e-01,
        2.22608695975035964e+00, -8.00245317566669279e-01,
        4.57031249277175089e-01, 2.18803419149470768e+00,
        -7.83003511263371976e-01, 4.64843748529596368e-01,
        2.15126051100659366e+00, -7.66053954531254355e-01,
        4.72656248830947701e-01, 2.11570248457175136e+00,
        -7.49386901356188240e-01, 4.80468748609962581e-01,
        2.08130081902951236e+00, -7.32993092000230995e-01,
        4.88281249241778237e-01, 2.04800000318021258e+00,
        -7.16863708730099525e-01, 4.96093748931098810e-01,
        2.01574803583926521e+00, -7.00990360175606675e-01,
        5.07812497779701388e-01, 1.96923077784079825e+00,
        -6.77642998396260410e-01, 5.23437498033319737e-01,
        1.91044776837204044e+00, -6.47337648285891021e-01,
        5.39062498006593560e-01, 1.85507247062801328e+00,
        -6.17923763020271188e-01, 5.54687498964024250e-01,
        1.80281690477552603e+00, -5.89350388745976339e-01,
        5.70312499806522322e-01, 1.75342465812909332e+00,
        -5.61570823110474571e-01, 5.85937497921867001e-01,
        1.70666667271966777e+00, -5.34542153929987052e-01,
        6.01562498226483444e-01, 1.66233766723853860e+00,
        -5.08224845014116688e-01, 6.17187498682654212e-01,
        1.62025316801528496e+00, -4.82582413587029357e-01,
        6.32812500000264566e-01, 1.58024691357958624e+00,
        -4.57581109246760320e-01, 6.48437499353274216e-01,
        1.54216867623689291e+00, -4.33189657120379490e-01,
        6.64062498728508976e-01, 1.50588235582451335e+00,
        -4.09379009344016609e-01, 6.79687498865382267e-01,
        1.47126437027210688e+00, -3.86122146934356092e-01,
        6.95312498728747119e-01, 1.43820224982050338e+00,
        -3.63393896015796081e-01, 7.10937499999943157e-01,
        1.40659340659351906e+00, -3.41170757402847080e-01,
        7.26562499999845568e-01, 1.37634408602179792e+00,
        -3.19430770766573779e-01, 7.42187500000120126e-01,
        1.34736842105241350e+00, -2.98153372318914478e-01,
        7.57812499999581890e-01, 1.31958762886670744e+00,
        -2.77319285416786077e-01, 7.73437498602746576e-01,
        1.29292929526503420e+00, -2.56910415591577124e-01,
        7.89062500000142664e-01, 1.26732673267303819e+00,
        -2.36909747078176913e-01, 8.04687500000259015e-01,
```



```
        1.24271844660154174e+00, -2.17301275689659512e-01,
        8.20312499999677036e-01, 1.21904761904809900e+00,
        -1.98069913762487504e-01, 8.35937499999997113e-01,
        1.19626168224299478e+00, -1.79201429457714445e-01,
        8.51562499999758749e-01, 1.17431192660583728e+00,
        -1.60682381690756770e-01, 8.67187500000204725e-01,
        1.15315315315288092e+00, -1.42500062607046951e-01,
        8.82812500000407896e-01, 1.13274336283133503e+00,
        -1.24642445206814556e-01, 8.98437499999816813e-01,
        1.11304347826109651e+00, -1.07098135556570995e-01,
        9.14062499999708455e-01, 1.09401709401744296e+00,
        -8.98563291221800009e-02, 9.29687500000063949e-01,
        1.07563025210076635e+00, -7.29067708080189947e-02,
        9.45312499999844014e-01, 1.05785123966959604e+00,
        -5.62397183230410880e-02, 9.60937500000120459e-01,
        1.04065040650393459e+00, -3.98459085470743157e-02,
        9.76562499999976685e-01, 1.02400000000002445e+00,
        -2.37165266173399170e-02, 9.92187500000169420e-01,
        1.00787401574785940e+00, -7.84317746085513856e-03,
        1.01562500000004907e+00, 9.84615384615337041e-01,
        1.55041865360135717e-02, 1.04687500000009237e+00,
        9.55223880596930641e-01, 4.58095360313824362e-02,
        1.07812500000002154e+00, 9.27536231884039442e-01,
        7.52234212376075018e-02, 1.10937499999982481e+00,
        9.01408450704367703e-01, 1.03796793681485644e-01,
        1.14062500000007416e+00, 8.76712328767066285e-01,
        1.31576357788784293e-01, 1.17187500000009659e+00,
        8.53333333333263000e-01, 1.58605030176721007e-01,
        1.20312499999950173e+00, 8.31168831169175393e-01,
        1.84922338493597849e-01, 1.23437500000022027e+00,
        8.10126582278336449e-01, 2.10564769107528083e-01,
        1.26562500000064615e+00, 7.90123456789720069e-01,
        2.35566071313277448e-01, 1.29687500000144706e+00,
        7.71084337348537208e-01, 2.59957524438041876e-01,
        1.32812499999945932e+00, 7.52941176470894757e-01,
        2.83768173130237500e-01, 1.35937500055846350e+00,
        7.35632183605830825e-01, 3.07025035705735583e-01,
        1.39062499999999467e+00, 7.19101123595508374e-01,
        3.29753286372464149e-01, 1.42187500000017564e+00,
        7.03296703296616421e-01, 3.51976423157301710e-01,
        1.45312500161088876e+00, 6.88172042247866766e-01,
        3.73716410902152685e-01, 1.48437500134602307e+00,
        6.73684209915422660e-01, 3.94993809147663466e-01,
        1.51562499999932343e+00, 6.59793814433284220e-01,
        4.15827895143264570e-01, 1.54687500000028200e+00,
        6.46464646464528614e-01, 4.36236766775100371e-01,
        1.57812500000061906e+00, 6.33663366336385092e-01,
        4.56237433481979870e-01, 1.60937500243255216e+00,
        6.21359222361793417e-01, 4.75845906381452632e-01,
        1.64062500000026312e+00, 6.09523809523711768e-01,
        4.95077266798011895e-01, 1.67187500000027911e+00,
        5.98130841121395473e-01, 5.13945751102401260e-01,
        1.70312500224662178e+00, 5.87155962528224662e-01,
        5.32464800188589216e-01, 1.73437500283893620e+00,
        5.76576575632799071e-01, 5.50647119589526390e-01,
        1.76562500399259092e+00, 5.66371680135198341e-01,
        5.68504737613959144e-01, 1.79687500443862880e+00,
        5.56521737755718449e-01, 5.86049047473771623e-01,
        1.82812500114411280e+00, 5.47008546666207462e-01,
        6.03290852063923744e-01, 1.85937500250667465e+00,
        5.37815125325376786e-01, 6.20240411099985067e-01,
        1.89062500504214515e+00, 5.28925618424108568e-01,
        6.36907464903988974e-01, 1.92187500371610143e+00,
        5.20325202245941476e-01, 6.53301273946326866e-01,
        1.95312500494870611e+00, 5.11999998702726389e-01,
        6.69430656476366792e-01, 1.98437500351688123e+00,
```

```
        5.03937006980894941e-01, 6.85304004871206018e-01,
        2.03125000000003997e+00, 4.92307692307682621e-01,
        7.08651367095930240e-01, 2.09375000579615866e+00,
        4.77611938976327366e-01, 7.38956719359554093e-01,
        2.15625000000061062e+00, 4.63768115941897652e-01,
        7.68370601797816022e-01, 2.21875000323311955e+00,
        4.50704224695355204e-01, 7.96943975698769513e-01,
        2.28125000853738547e+00, 4.38356162743050726e-01,
        8.24723542091080120e-01, 2.34374999999916556e+00,
        4.26666666666818573e-01, 8.51752210736227866e-01,
        2.40625000438447856e+00, 4.15584414827170512e-01,
        8.78069520876078258e-01, 2.46875000884389584e+00,
        4.05063289688167072e-01, 9.03711953249632494e-01,
        2.53124999999940403e+00, 3.95061728395154743e-01,
        9.28713251872476775e-01, 2.59375000434366632e+00,
        3.85542168029044230e-01, 9.53104706671537905e-01,
        2.65625000734081196e+00, 3.76470587194880080e-01,
        9.76915356454189698e-01, 2.71875000787161980e+00,
        3.67816090889081959e-01, 1.00017221875016560e+00,
        2.78125001557333462e+00, 3.59550559784484969e-01,
        1.02290047253181449e+00, 2.84375001147093220e+00,
        3.51648350229895601e-01, 1.04512360775085789e+00,
        2.90625000771072894e+00, 3.44086020592463127e-01,
        1.06686359300668343e+00, 2.96875001371853831e+00,
        3.36842103706616824e-01, 1.08814099342179560e+00,
        3.03125000512624965e+00, 3.29896906658595002e-01,
        1.10897507739479018e+00, 3.09375001373132807e+00,
        3.23232321797685962e-01, 1.12938395177327244e+00,
        3.15625001204422961e+00, 3.16831681959289180e-01,
        1.14938461785752644e+00, 3.21875000888250318e+00,
        3.10679610793130057e-01, 1.16899308818952186e+00,
        3.28125000000102052e+00, 3.04761904761809976e-01,
        1.18822444735810784e+00, 3.34375001587649123e+00,
        2.99065419140752298e-01, 1.20709293641028914e+00,
        3.40625000791328070e+00, 2.93577980969346064e-01,
        1.22561198175258212e+00, 3.46875000615970519e+00,
        2.88288287776354346e-01, 1.24379430028837845e+00,
        3.53125000516822674e+00, 2.83185840293502689e-01,
        1.26165191737618265e+00, 3.59375001425228779e+00,
        2.78260868461675415e-01, 1.27919622952937750e+00,
        3.65625001719730669e+00, 2.73504272217836075e-01,
        1.29643803670156643e+00, 3.71875000856489324e+00,
        2.68907562405871714e-01, 1.31338759261496740e+00,
        3.78125001788371806e+00, 2.64462808666557803e-01,
        1.33005464752659286e+00, 3.84375001532508964e+00,
        2.60162600588744020e-01, 1.34644845655970613e+00,
        3.90625000429340918e+00, 2.55999999718627136e-01,
        1.36257783560168733e+00, 3.96875001912740766e+00,
        2.51968502722644594e-01, 1.37845118847836900e+00,
        4.06250002536431332e+00, 2.46153844616978895e-01,
        1.40179855389937913e+00, 4.18750001743208244e+00,
        2.38805969155131859e-01, 1.43210390131407017e+00,
        4.31250002253733200e+00, 2.31884056759177282e-01,
        1.46151778758352613e+00, 4.43750000671406397e+00,
        2.25352112335092170e-01, 1.49009115631456268e+00,
        4.56250002627485340e+00, 2.19178080929562313e-01,
        1.51787072466748185e+00, 4.68750001185115028e+00,
        2.13333327793974317e-01, 1.54489939382477459e+00,
        4.81250001682742301e+00, 2.07792207065640028e-01,
        1.57121670311050998e+00, 4.93750000000042366e+00,
        2.02531645569602875e-01, 1.59685913022732606e+00,
        5.06249999999927613e+00, 1.97530864197559108e-01,
        1.62186043243251454e+00, 5.18750002327641901e+00,
        1.92771083472381588e-01, 1.64625189004383721e+00,
        5.31250002381002329e+00, 1.88235293273997795e-01,
        1.67006253873242194e+00, 5.43750000000577405e+00,
```



```
        1.83908045976816203e-01, 1.69331939641586438e+00,
        5.56250002193114934e+00, 1.79775280190080267e-01,
        1.71604765143503712e+00, 5.68749999999938005e+00,
        1.75824175824194989e-01, 1.73827078427695980e+00,
        5.81250002749782002e+00, 1.72043009938785768e-01,
        1.76001077564428243e+00, 5.93749999999874767e+00,
        1.68421052631614471e-01, 1.78128816936054868e+00,
        6.06250001966917473e+00, 1.64948453073088669e-01,
        1.80212225950800153e+00, 6.18750003004243609e+00,
        1.61616160831459688e-01, 1.82253113275015188e+00,
        6.31250002448351388e+00, 1.58415840969730465e-01,
        1.84253179848005466e+00, 6.43750001359968849e+00,
        1.55339805497076044e-01, 1.86214026810242750e+00,
        6.56250003345742350e+00, 1.52380951604072529e-01,
        1.88137163301601618e+00, 6.68750002403557531e+00,
        1.49532709742937614e-01, 1.90024011581622965e+00,
        6.81250003423489581e+00, 1.46788990088028509e-01,
        1.91875916501466826e+00, 6.93750003062940923e+00,
        1.44144143507740546e-01, 1.93694148348760287e+00,
        7.06250002747386052e+00, 1.41592919803171097e-01,
        1.95479910036266347e+00, 7.18750003617887856e+00,
        1.39130434082284093e-01, 1.97234341115705192e+00,
        7.31250000000050537e+00, 1.36752136752127301e-01,
        1.98958521255804399e+00, 7.43750002212249761e+00,
        1.34453781112678528e-01, 2.00653477384620160e+00,
        7.56250003604752941e+00, 1.32231404328381430e-01,
        2.02320182812357530e+00, 7.68750005007207449e+00,
        1.30081299965731312e-01, 2.03959563964607682e+00,
        7.81249996125652668e+00, 1.28000000634773070e-01,
        2.05572501010335529e+00, 7.93750005224239974e+00,
        1.25984251139310915e-01, 2.07159837080052966e+00,
        8.12500004244456164e+00, 1.23076922433975874e-01,
        2.09494573343974722e+00, 8.37500006149772425e+00,
        1.19402984197849338e-01, 2.12525108505414195e+00,
        8.62500006593247370e+00, 1.15942028099206410e-01,
        2.15466497056176820e+00, 8.87500007743793873e+00,
        1.12676055354884341e-01, 2.18323834408688100e+00,
        9.12500001754142609e+00, 1.09589040885222130e-01,
        2.21101790139090326e+00, 9.37500007707016181e+00,
        1.06666665789779500e-01, 2.23804658007729174e+00,
        9.62500004426353151e+00, 1.03896103418305616e-01,
        2.26436388477265638e+00, 9.87500006518495788e+00,
        1.01265822116353585e-01, 2.29000631738819393e+00,
        1.01250000000026539e+01, 9.87654320987395445e-02,
        2.31500761299286495e+00, 1.03750000409819823e+01,
        9.63855417879450060e-02, 2.33939907006683256e+00,
        1.06250000362555337e+01, 9.41176467376672460e-02,
        2.36320971822276604e+00, 1.08750000879032314e+01,
        9.19540222452362582e-02, 2.38646658505780351e+00,
        1.11250000697274576e+01, 8.98876398860551373e-02,
        2.40919483431994053e+00, 1.13750000462194141e+01,
        8.79120875548795450e-02, 2.43141796890025930e+00,
        1.16250000714972366e+01, 8.60215048472860316e-02,
        2.45315795762371991e+00, 1.18750000788855150e+01,
        8.42105257563797310e-02, 2.47443535656369562e+00,
        1.21250000895724916e+01, 8.24742261948517991e-02,
        2.49526944421096886e+00, 1.23750000985058719e+01,
        8.08080801648427965e-02, 2.51567831641482442e+00,
        1.26250000894226950e+01, 7.92079202310506381e-02,
        2.53567898224440924e+00, 1.28750000768594433e+01,
        7.76699024489580225e-02, 2.55528745251946532e+00,
        1.31250000578007420e+01, 7.61904758549435401e-02,
        2.57451881288155349e+00, 1.33750000809310077e+01,
        7.47663546877819496e-02, 2.59338729883298669e+00,
        1.36250000915049636e+01, 7.33944949199294983e-02,
        2.61190634726526838e+00, 1.38750000830616607e+01,
```

```
        7.20720716406179490e-02, 2.63008866561892418e+00,
        1.41249999999960103e+01, 7.07964601770111474e-02,
        2.64794627703222218e+00, 1.43750000290097564e+01,
        6.95652172509168693e-02, 2.66549058870148414e+00,
        1.46250000868097665e+01, 6.83760679702078294e-02,
        2.68273239905363070e+00, 1.48750000966053975e+01,
        6.72268903196987927e-02, 2.69968195792617394e+00,
        1.51250001097012756e+01, 6.61157019998031836e-02,
        2.71634901116988203e+00, 1.53750000510427132e+01,
        6.50406501905787804e-02, 2.73274281701243282e+00,
        1.56250001080665442e+01, 6.39999995573594382e-02,
        2.74887220253872400e+00, 1.58750000434989929e+01,
        6.29921258116476201e-02, 2.76474554751884938e+00,
        1.62500000641781739e+01, 6.15384612954199342e-02,
        2.78809291272517257e+00, 1.67500001015987401e+01,
        5.97014921751882754e-02, 2.81839826433667184e+00,
        1.72500001048300184e+01, 5.79710141404578272e-02,
        2.84781214955447126e+00, 1.77500001262529885e+01,
        5.63380277682904579e-02, 2.87638552303426920e+00,
        1.82500001543340602e+01, 5.47945200845665337e-02,
        2.90416508848516131e+00, 1.87500001096404212e+01,
        5.33333330214672482e-02, 2.93119375826390893e+00,
        1.92500001680268191e+01, 5.19480514946147609e-02,
        2.95751106946245912e+00, 1.97500000329124035e+01,
        5.06329113080278073e-02, 2.98315349301358168e+00,
        2.02500001270002485e+01, 4.93827157396732261e-02,
        3.00815479982416534e+00, 2.07500001519906796e+01,
        4.81927707313324349e-02, 3.03254625400155930e+00,
        2.12500001425219267e+01, 4.70588232137922752e-02,
        3.05635690207734001e+00, 2.17500000758314478e+01,
        4.59770113339538697e-02, 3.07961376102119644e+00,
        2.22500001767207358e+01, 4.49438198677525880e-02,
        3.10234201655475417e+00, 2.27500001365873317e+01,
        4.39560436921389575e-02, 3.12456515140079816e+00,
        2.32500001697599998e+01, 4.30107523741288036e-02,
        3.14630513933487066e+00, 2.37500001766865303e+01,
        4.21052628446554611e-02, 3.16758253792008304e+00,
 25 9.47265623608246343e-02, 1.05567010464380857e+01, -2.35676082856530300e+00,
 26 9.66796869131412717e-02, 1.03434344062203838e+01, -2.33635196153499791e+00,
 27 9.86328118117651004e-02, 1.01386139321308306e+01, -2.31635129573594156e+00,
 28 1.00585936733578435e-01, 9.94174764856737347e+00, -2.29674282498938709e+00,
 29 1.02539062499949152e-01, 9.75238095238578850e+00, -2.27751145544242561e+00,
 30 1.04492186859904843e-01, 9.57009351656812157e+00, -2.25864297726331742e+00,
 31 1.06445312294918631e-01, 9.39449543094380957e+00, -2.24012392529694537e+00,
 32 1.08398437050250693e-01, 9.22522526350104144e+00, -2.22194160843615762e+00,
 33 1.10351562442130582e-01, 9.06194690740703912e+00, -2.20408398741152212e+00,
 34 1.12304686894746625e-01, 8.90434787407592943e+00, -2.18653968262558962e+00,
 35 1.14257811990227776e-01, 8.75213679118525256e+00, -2.16929787526329321e+00,
 36 1.16210936696872255e-01, 8.60504207627572093e+00, -2.15234831939887172e+00,
 37 1.18164061975360682e-01, 8.46280995492959498e+00, -2.13568126444263484e+00,
 38 1.20117187499996322e-01, 8.32520325203277523e+00, -2.11928745022706622e+00,
 39 1.22070312499895098e-01, 8.19200000000703987e+00, -2.10315806829801133e+00,
 40 1.24023436774175100e-01, 8.06299217317146599e+00, -2.08728472499318229e+00,
 41 1.26953123746900931e-01, 7.87692315467275872e+00, -2.06393736501443570e+00,
 42 1.30859374098123454e-01, 7.64179109744297769e+00, -2.03363201254049386e+00,
 43 1.34765623780674720e-01, 7.42028992220936967e+00, -2.00421812948999545e+00,
 44 1.38671874242985771e-01, 7.21126764500034501e+00, -1.97564475345722457e+00,
 45 1.42578124148616536e-01, 7.01369867201821506e+00, -1.94786518986246371e+00,
 46 1.46484374166731490e-01, 6.82666670549979404e+00, -1.92083651719164372e+00,
 47 1.50390624434435488e-01, 6.64935067435644189e+00, -1.89451920694646070e+00,
 48 1.54296874339723084e-01, 6.48101268596180624e+00, -1.86887677685174936e+00,
 49 1.58203124999987427e-01, 6.32098765432149001e+00, -1.84387547036714849e+00,
 50 1.62109374999815342e-01, 6.16867469880220742e+00, -1.81948401724404896e+00,
 51 1.66015624243955634e-01, 6.02352943919619310e+00, -1.79567337310324682e+00,
 52 1.69921874302298687e-01, 5.88505749542848644e+00, -1.77241651049093640e+00,
 53 1.73828124315277527e-01, 5.75280901142480605e+00, -1.74968825924644555e+00,
```



```
 54 1.77734374286237506e-01, 5.62637364896854919e+00, -1.72746512253855222e+00,
 55 1.81640624146994889e-01, 5.50537636993989743e+00, -1.70572513658236602e+00,
 56 1.85546874316304788e-01, 5.38947370406942916e+00, -1.68444773712372431e+00,
 57 1.89453124405085355e-01, 5.27835053203882509e+00, -1.66361364967629299e+00,
 58 1.93359374570531595e-01, 5.17171718320401652e+00, -1.64320477712600699e+00,
 59 1.97265624263334577e-01, 5.06930694962380368e+00, -1.62320411193263148e+00,
 60 2.01171874086291030e-01, 4.97087380898513764e+00, -1.60359564135180399e+00,
 61 2.05078123979995308e-01, 4.87619050044336610e+00, -1.58436427985572159e+00,
 62 2.08984373896073439e-01, 4.78504675424820736e+00, -1.56549579585994181e+00,
 63 2.12890623963011144e-01, 4.69724772930228163e+00, -1.54697674768135762e+00,
 64 2.16796874723889421e-01, 4.61261261848719517e+00, -1.52879442500076479e+00,
 65 2.20703124198150608e-01, 4.53097346778917753e+00, -1.51093680996032553e+00,
 66 2.24609374375627030e-01, 4.45217392541970725e+00, -1.49339249945607477e+00,
 67 2.28515625000094036e-01, 4.37606837606657528e+00, -1.47615069024134016e+00,
 68 2.32421873924349737e-01, 4.30252102831546246e+00, -1.45920113655598627e+00,
 69 2.36328123935216378e-01, 4.23140497774241098e+00, -1.44253408394829741e+00,
 70 2.40234375000066919e-01, 4.16260162601510064e+00, -1.42614026966681173e+00,
 71 2.44140623863132178e-01, 4.09600001907347711e+00, -1.41001089239381727e+00,
 72 2.48046874999894917e-01, 4.03149606299383390e+00, -1.39413753858134015e+00,
 73 2.53906248590769879e-01, 3.93846156032078243e+00, -1.37079018013412401e+00,
 74 2.61718748558906533e-01, 3.82089554342693294e+00, -1.34048483059486401e+00,
 75 2.69531249159214337e-01, 3.71014493910979404e+00, -1.31107094300173976e+00,
 76 2.77343749428383191e-01, 3.60563381024826013e+00, -1.28249756949928795e+00,
 77 2.85156249289339359e-01, 3.50684932380819214e+00, -1.25471800582335113e+00,
 78 2.92968749999700462e-01, 3.41333333333682321e+00, -1.22768933094427446e+00,
 79 3.00781248554318814e-01, 3.32467534065511261e+00, -1.20137202743229921e+00,
 80 3.08593748521894806e-01, 3.24050634463533127e+00, -1.17572959680235023e+00,
 81 3.16406249999639899e-01, 3.16049382716409077e+00, -1.15072828980826181e+00,
 82 3.24218749999785061e-01, 3.08433734939963511e+00, -1.12633683668362750e+00,
 83 3.32031248841858584e-01, 3.01176471638753718e+00, -1.10252619147729547e+00,
 84 3.39843749265406558e-01, 2.94252874199264314e+00, -1.07929632798654107e+00,
 85 3.47656249999834799e-01, 2.87640449438338930e+00, -1.05654107474789782e+00,
 86 3.55468749999899247e-01, 2.81318681318761055e+00, -1.03431793796299587e+00,
 87 3.63281249999864997e-01, 2.75268817204403371e+00, -1.01257795132667816e+00,
 88 3.71093749064121570e-01, 2.69473684890124421e+00, -9.91300555400967731e-01,
 89 3.78906249999751032e-01, 2.63917525773369288e+00, -9.70466465976836723e-01,
 90 3.86718748879039009e-01, 2.58585859335407608e+00, -9.50057597243619156e-01,
 91 3.94531249999987899e-01, 2.53465346534661240e+00, -9.30056927638333697e-01,
 92 4.02343749999485523e-01, 2.48543689320706163e+00, -9.10448456251205407e-01,
 93 4.10156249999862586e-01, 2.43809524059864202e+00, -8.91217095348825872e-01,
 94 4.17968749447214571e-01, 2.39252336765021800e+00, -8.72348611340208357e-01,
 95 4.25781248601723117e-01, 2.34862386092395203e+00, -8.53829565534445223e-01,
 96 4.33593749393073047e-01, 2.30630630953458038e+00, -8.35647244566987801e-01,
 97 4.41406248572254134e-01, 2.26548673299152270e+00, -8.17789629001761220e-01,
 98 4.49218749348472501e-01, 2.22608695975035964e+00, -8.00245317566669279e-01,
 99 4.57031249277175089e-01, 2.18803419149470768e+00, -7.83003511263371976e-01,
100 4.64843748529596368e-01, 2.15126051100659366e+00, -7.66053954531254355e-01,
101 4.72656248830947701e-01, 2.11570248457175136e+00, -7.49386901356188240e-01,
102 4.80468748609962581e-01, 2.08130081902951236e+00, -7.32993092000230995e-01,
103 4.88281249241778237e-01, 2.04800000318021258e+00, -7.16863708730099525e-01,
104 4.96093748931098810e-01, 2.01574803583926521e+00, -7.00990360175606675e-01,
105 5.07812497779701388e-01, 1.96923077784079825e+00, -6.77642998396260410e-01,
106 5.23437498033319737e-01, 1.91044776837204044e+00, -6.47337648285891021e-01,
107 5.39062498006593560e-01, 1.85507247062801328e+00, -6.17923763020271188e-01,
108 5.54687498964024250e-01, 1.80281690477552603e+00, -5.89350388745976339e-01,
109 5.70312499806522322e-01, 1.75342465812909332e+00, -5.61570823110474571e-01,
110 5.85937497921867001e-01, 1.70666667271966777e+00, -5.34542153929987052e-01,
111 6.01562498226483444e-01, 1.66233766723853860e+00, -5.08224845014116688e-01,
112 6.17187498682654212e-01, 1.62025316801528496e+00, -4.82582413587029357e-01,
113 6.32812500000264566e-01, 1.58024691357958624e+00, -4.57581109246760320e-01,
114 6.48437499353274216e-01, 1.54216867623689291e+00, -4.33189657120379490e-01,
115 6.64062498728508976e-01, 1.50588235582451335e+00, -4.09379009344016609e-01,
116 6.79687498865382267e-01, 1.47126437027210688e+00, -3.86122146934356092e-01,
117 6.95312498728747119e-01, 1.43820224982050338e+00, -3.63393896015796081e-01,
118 7.10937499999943157e-01, 1.40659340659351906e+00, -3.41170757402847080e-01,
119 7.26562499999845568e-01, 1.37634408602179792e+00, -3.19430770766573779e-01,
```

```
7.42187500000120126e-01, 1.34736842105241350e+00, -2.98153372318914478e-01,
7.57812499999581890e-01, 1.31958762886670744e+00, -2.77319285416786077e-01,
7.73437498602746576e-01, 1.29292929526503420e+00, -2.56910415591577124e-01,
7.89062500000142664e-01, 1.26732673267303819e+00, -2.36909747078176913e-01,
8.04687500000259015e-01, 1.24271844660154174e+00, -2.17301275689659512e-01,
8.20312499999677036e-01, 1.21904761904809900e+00, -1.98069913762487504e-01,
8.35937499999997113e-01, 1.19626168224299478e+00, -1.79201429457714445e-01,
8.51562499999758749e-01, 1.17431192660583728e+00, -1.60682381690756770e-01,
8.67187500000204725e-01, 1.15315315315288092e+00, -1.42500062607046951e-01,
8.82812500000407896e-01, 1.13274336283133503e+00, -1.24642445206814556e-01,
8.98437499999816813e-01, 1.11304347826109651e+00, -1.07098135556570995e-01,
9.14062499999708455e-01, 1.09401709401744296e+00, -8.98563291221800009e-02,
9.29687500000063949e-01, 1.07563025210076635e+00, -7.29067708080189947e-02,
9.45312499999844014e-01, 1.05785123966959604e+00, -5.62397183230410880e-02,
9.60937500000120459e-01, 1.04065040650393459e+00, -3.98459085470743157e-02,
9.76562499999976685e-01, 1.02400000000002445e+00, -2.37165266173399170e-02,
9.92187500000169420e-01, 1.00787401574785940e+00, -7.84317746085513856e-03,
1.01562500000004907e+00, 9.84615384615337041e-01,  1.55041865360135717e-02,
1.04687500000009237e+00, 9.55223880596930641e-01,  4.58095360313824362e-02,
1.07812500000002154e+00, 9.27536231884039442e-01,  7.52234212376075018e-02,
1.10937499999982481e+00, 9.01408450704367703e-01,  1.03796793681485644e-01,
1.14062500000007416e+00, 8.76712328767066285e-01,  1.31576357788784293e-01,
1.17187500000009659e+00, 8.53333333333263000e-01,  1.58605030176721007e-01,
1.20312499999950173e+00, 8.31168831169175393e-01,  1.84922338493597849e-01,
1.23437500000022027e+00, 8.10126582278336449e-01,  2.10564769107528083e-01,
1.26562500000064615e+00, 7.90123456789720069e-01,  2.35566071313277448e-01,
1.29687500000144706e+00, 7.71084337348537208e-01,  2.59957524438041876e-01,
1.32812499999945932e+00, 7.52941176470894757e-01,  2.83768171302375000e-01,
1.35937500055846350e+00, 7.35632183605830825e-01,  3.07025035705735583e-01,
1.39062499999999467e+00, 7.19101123595508374e-01,  3.29753286372464149e-01,
1.42187500000017564e+00, 7.03296703296616421e-01,  3.51976423157301710e-01,
1.45312500161088876e+00, 6.88172042247866766e-01,  3.73716410902152685e-01,
1.48437500134602307e+00, 6.73684209915422660e-01,  3.94993809147663466e-01,
1.51562499999932343e+00, 6.59793814433284220e-01,  4.15827895143264570e-01,
1.54687500000028200e+00, 6.46464646464528614e-01,  4.36236766775100371e-01,
1.57812500000061906e+00, 6.33663366336385092e-01,  4.56237433481979870e-01,
1.60937500243255216e+00, 6.21359223361793417e-01,  4.75845906381452632e-01,
1.64062500000026312e+00, 6.09523809523711768e-01,  4.95077266798011895e-01,
1.67187500000027911e+00, 5.98130841121395473e-01,  5.13945751102401260e-01,
1.70312500224662178e+00, 5.87155962528246662e-01,  5.32464801885892216e-01,
1.73437500283893620e+00, 5.76576575632799071e-01,  5.50647119589526390e-01,
1.76562500399259092e+00, 5.66371680135198341e-01,  5.68504737613959144e-01,
1.79687500443862880e+00, 5.56521737755718449e-01,  5.86049047473771623e-01,
1.82812500114411280e+00, 5.47008546666207462e-01,  6.03290852063923744e-01,
1.85937500250667465e+00, 5.37815125325376786e-01,  6.20240411099985067e-01,
1.89062500504214515e+00, 5.28925618424108568e-01,  6.36907464903988974e-01,
1.92187500371610143e+00, 5.20325202245941476e-01,  6.53301273946326866e-01,
1.95312500494870611e+00, 5.11999998702726389e-01,  6.69430656476366792e-01,
1.98437500351688123e+00, 5.03937006980894941e-01,  6.85304004871206018e-01,
2.03125000000003997e+00, 4.92307692307682621e-01,  7.08651367095930240e-01,
2.09375000579615866e+00, 4.77611938976327366e-01,  7.38956719359554093e-01,
2.15625000000061062e+00, 4.63768115941897652e-01,  7.68370601797816022e-01,
2.21875000323311955e+00, 4.50704224695355204e-01,  7.96943975698769513e-01,
2.28125000853738547e+00, 4.38356162743050726e-01,  8.24723542091080120e-01,
2.34374999999916556e+00, 4.26666666666818573e-01,  8.51752210736227866e-01,
2.40625000438447856e+00, 4.15584414827170512e-01,  8.78069520876078258e-01,
2.46875000884389584e+00, 4.05063289688167072e-01,  9.03711953249632494e-01,
2.53124999999940403e+00, 3.95061728395154743e-01,  9.28713251872476775e-01,
2.59375000434366632e+00, 3.85542168029044230e-01,  9.53104706671537905e-01,
2.65625000734081196e+00, 3.76470587194880080e-01,  9.76915356454189698e-01,
2.71875000787161980e+00, 3.67816090889081959e-01,  1.00017221875016560e+00,
2.78125001557333462e+00, 3.59550559784484969e-01,  1.02290047253181449e+00,
2.84375001147093220e+00, 3.51648350229895601e-01,  1.04512360775085789e+00,
2.90625000771072894e+00, 3.44086205924631227e-01,  1.06686359300668343e+00,
2.96875001371853831e+00, 3.36842103706616824e-01,  1.08814099342179560e+00,
3.03125000512624965e+00, 3.29896906658595002e-01,  1.10897507739479018e+00,
```



```
3.09375001373132807e+00, 3.23232321797685962e-01,  1.12938395177327244e+00,
3.15625001204422961e+00, 3.16831681959289180e-01,  1.14938461785752644e+00,
3.21875000888250318e+00, 3.10679610793130057e-01,  1.16899308818952186e+00,
3.28125000000102052e+00, 3.04761904761809976e-01,  1.18822444735810784e+00,
3.34375001587649123e+00, 2.99065419140752298e-01,  1.20709293641028914e+00,
3.40625000791328070e+00, 2.93577980969346064e-01,  1.22561198175258212e+00,
3.46875000615970519e+00, 2.88288287776354346e-01,  1.24379430028837845e+00,
3.53125000516822674e+00, 2.83185840293502689e-01,  1.26165191737618265e+00,
3.59375001425228779e+00, 2.78260868461675415e-01,  1.27919622952937750e+00,
3.65625001719730669e+00, 2.73504722217836075e-01,  1.29643803670156643e+00,
3.71875000856489324e+00, 2.68907562405871714e-01,  1.31338759261496740e+00,
3.78125001788371806e+00, 2.64462808666557803e-01,  1.33005464752659286e+00,
3.84375001532508964e+00, 2.60162600588744020e-01,  1.34644845655970613e+00,
3.90625000429340918e+00, 2.55999999718627136e-01,  1.36257783560168733e+00,
3.96875001912740766e+00, 2.51968502722644594e-01,  1.37845118847836900e+00,
4.06250002536431332e+00, 2.46153844616978895e-01,  1.40179855389937913e+00,
4.18750001743208244e+00, 2.38805969155131859e-01,  1.43210390131407017e+00,
4.31250002253733200e+00, 2.31884056759177282e-01,  1.46151778758352613e+00,
4.43750000671406397e+00, 2.25352112335092170e-01,  1.49009115631456268e+00,
4.56250002627485340e+00, 2.19178080929562313e-01,  1.51787072466748185e+00,
4.68750001185115028e+00, 2.13333332793974317e-01,  1.54489939382477459e+00,
4.81250001682742301e+00, 2.07792207065640028e-01,  1.57121670311050998e+00,
4.93750000000042366e+00, 2.02531645569602875e-01,  1.59685913022732606e+00,
5.06249999999927613e+00, 1.97530864197559108e-01,  1.62186043243251454e+00,
5.18750002327641901e+00, 1.92771084372381588e-01,  1.64625189004383721e+00,
5.31250002381002329e+00, 1.88235293273997795e-01,  1.67006253873242194e+00,
5.43750000000577405e+00, 1.83908045976816203e-01,  1.69331939641586438e+00,
5.56250002193114934e+00, 1.79775280190080267e-01,  1.71604765143503712e+00,
5.68749999999938005e+00, 1.75824175824194989e-01,  1.73827078427695980e+00,
5.81250002749782002e+00, 1.72043009938785768e-01,  1.76001077564428243e+00,
5.93749999999874767e+00, 1.68421052631614471e-01,  1.78128816936054868e+00,
6.06250001966917473e+00, 1.64948453073088669e-01,  1.80212225950800153e+00,
6.31250002448351388e+00, 1.58415840969730465e-01,  1.84253179848005466e+00,
6.43750001359968849e+00, 1.55339805497076044e-01,  1.86214026810242750e+00,
6.56250003345742350e+00, 1.52380951604072529e-01,  1.88137163301601618e+00,
6.68750002403557531e+00, 1.49532709742937614e-01,  1.90024011581622965e+00,
6.81250003423489581e+00, 1.46788990088028509e-01,  1.91875916501466826e+00,
6.93750003062940923e+00, 1.44144143507740546e-01,  1.93694148348760287e+00,
7.06250002747386052e+00, 1.41592919803171097e-01,  1.95479910036266347e+00,
7.18750003617887856e+00, 1.39130434082284093e-01,  1.97234341115705192e+00,
7.31250000000050537e+00, 1.36752136752127301e-01,  1.98958521255804399e+00,
7.43750002212249761e+00, 1.34453781112678528e-01,  2.00653477384620160e+00,
7.56250003604752941e+00, 1.32231404328381430e-01,  2.02320182812357530e+00,
7.68750005007207449e+00, 1.30081299965731312e-01,  2.03959563964607682e+00,
7.81249996125652668e+00, 1.28000000634773070e-01,  2.05572501010335529e+00,
7.93750005224239974e+00, 1.25984251139310915e-01,  2.07159837080052966e+00,
8.12500004244456164e+00, 1.23076922433975874e-01,  2.09494573343974722e+00,
8.37500001754142609e+00, 1.19402981497849338e-01,  2.12525108505414195e+00,
8.62500006593247370e+00, 1.15942028099206410e-01,  2.15466497056176820e+00,
8.87500007743793873e+00, 1.12676055354884341e-01,  2.18323834408688100e+00,
9.12500001754142609e+00, 1.09589040885222130e-01,  2.21101790139090326e+00,
9.37500007707016181e+00, 1.06666665789779500e-01,  2.23804658007729174e+00,
9.62500004426353151e+00, 1.03896103418305616e-01,  2.26436388477265638e+00,
9.87500006518495788e+00, 1.01265822116353585e-01,  2.29000631738819393e+00,
1.01250000000026539e+01, 9.87654320987395445e-02,  2.31500761299286495e+00,
1.03750000409819823e+01, 9.63855417879450060e-02,  2.33939907006683256e+00,
1.06250000362555337e+01, 9.41176467376672460e-02,  2.36320971822276604e+00,
1.08750000879032314e+01, 9.19540222452362582e-02,  2.38646658505780351e+00,
1.11250000697274576e+01, 8.98876398860551373e-02,  2.40919483431994053e+00,
1.13750000462194141e+01, 8.79120875548795450e-02,  2.43141796890025930e+00,
1.16250000714972366e+01, 8.60215048472860316e-02,  2.45315795762371991e+00,
1.18750000788855150e+01, 8.42105257563797310e-02,  2.47443535656369562e+00,
1.21250000895724916e+01, 8.24742261948517991e-02,  2.49526944421096886e+00,
1.23750000985058719e+01, 8.08080801648427965e-02,  2.51567831641482442e+00,
1.26250000894226950e+01, 7.92079202310506381e-02,  2.53567898224440924e+00,
```

```
1.28750000768594433e+01, 7.76699024489580225e-02,  2.55528745251946532e+00,
1.31250000578007420e+01, 7.61904758549435401e-02,  2.57451881288155349e+00,
1.33750000809310077e+01, 7.47663546877819496e-02,  2.59338729883298669e+00,
1.36250000915049636e+01, 7.33944949199294983e-02,  2.61190634726526838e+00,
1.38750000830616607e+01, 7.20720716406179490e-02,  2.63008866561892418e+00,
1.41249999999960103e+01, 7.07964601770111474e-02,  2.64794627703222218e+00,
1.43750000290097564e+01, 6.95652172509168693e-02,  2.66549058870148414e+00,
1.46250000868097665e+01, 6.83760679702078294e-02,  2.68273239905363070e+00,
1.48750000966053975e+01, 6.72268903196987927e-02,  2.69968195792617394e+00,
1.51250001097012756e+01, 6.61157019998031836e-02,  2.71634901116988203e+00,
1.53750000510427132e+01, 6.50406501905787804e-02,  2.73274281701243282e+00,
1.56250001080665442e+01, 6.39999995573594382e-02,  2.74887220253872400e+00,
1.58750000434989929e+01, 6.29921258116476201e-02,  2.76474554751884938e+00,
1.62500000641781739e+01, 6.15384612954199342e-02,  2.78809291272517257e+00,
1.67500001015987401e+01, 5.97014921751882754e-02,  2.81839826433667184e+00,
1.72500001048300184e+01, 5.79710141404578272e-02,  2.84781214955447126e+00,
1.77500001262529885e+01, 5.63380277682904579e-02,  2.87638552303426920e+00,
1.82500001543340602e+01, 5.47945200845665337e-02,  2.90416508848516131e+00,
1.87500001096404212e+01, 5.33333330214672482e-02,  2.93119375826390893e+00,
1.92500001680268191e+01, 5.19480514946147609e-02,  2.95751106946245912e+00,
1.97500000329124035e+01, 5.06329113080278073e-02,  2.98315349301358168e+00,
2.02500001270002485e+01, 4.93827157396732261e-02,  3.00815479982416534e+00,
2.07500001519906796e+01, 4.81927707313324349e-02,  3.03254625400155930e+00,
2.12500001425219267e+01, 4.70588232137922752e-02,  3.05635690207734001e+00,
2.17500000758314478e+01, 4.59770113339538697e-02,  3.07961376102119644e+00,
2.22500001767207358e+01, 4.49438198677525880e-02,  3.10234201655475417e+00,
2.27500001365873317e+01, 4.39560436921389575e-02,  3.12456515140079816e+00,
2.32500001697599998e+01, 4.30107523741288036e-02,  3.14630513933487066e+00,
2.37500001766865303e+01, 4.21052628446554611e-02,  3.16758253792008304e+00,
};
```

*_____unchanged_portion_omitted_*

```
**********************************************************
    7623 Thu Jun  6 20:37:31 2019
new/usr/src/lib/libm/common/C/_TBL_log2.c
11210 libm should be cstyle(10NBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#include "libm_protos.h"

const double _TBL_log2_hi[] = {
        0.00000000000000000e+00, 1.12272500991821289e-02,
        2.23678052425384521e-02, 3.34229767322540283e-02,
        4.43941056728363037e-02, 5.52824139595031738e-02,
        6.60891532897949219e-02, 7.68155455589294434e-02,
        8.74627828598022461e-02, 9.80320572853088379e-02,
        1.08524441719055176e-01, 1.18941068649291992e-01,
        1.29282951354980469e-01, 1.39551281929016113e-01,
        1.49747014045715332e-01, 1.59871220588684082e-01,
        1.69924974441528320e-01, 1.79908990859985352e-01,
        1.89824461936950684e-01, 1.99672341346740723e-01,
        2.09453344345092773e-01, 2.19168424606323242e-01,
        2.28818655014038086e-01, 2.38404631614685059e-01,
        2.47927427291870117e-01, 2.57387638092041016e-01,
        2.66786336898803711e-01, 2.76124238967895508e-01,
        2.85402059555053711e-01, 2.94620513916015625e-01,
        3.03780555725097656e-01, 3.12882900238037109e-01,
        3.21928024291992188e-01, 3.30916643142700195e-01,
        3.39849948883056641e-01, 3.48727941513061523e-01,
        3.57551813125610352e-01, 3.66322040557861328e-01,
        3.75039339065551758e-01, 3.83704185485839844e-01,
        3.92317295074462891e-01, 4.00879383087158203e-01,
        4.09309263610839984e-01, 4.17852401733398438e-01,
        4.26264524459838867e-01, 4.34628009796142578e-01,
        4.42943334579467773e-01, 4.51210975646972656e-01,
        4.59431409835815430e-01, 4.67605352401733398e-01,
        4.75733280181884766e-01, 4.83815670013427734e-01,
        4.91852998733520508e-01, 4.99845743179321289e-01,
```



```
        5.07794380187988281e-01, 5.15699386596679688e-01,
        5.23561954498291016e-01, 5.31381130218505859e-01,
        5.39158344268798828e-01, 5.46894073486328125e-01,
        5.54588794708251953e-01, 5.62242031097412109e-01,
        5.69855213165283203e-01, 5.77428817749023438e-01,
        5.84962368011474609e-01, 5.92456817626953125e-01,
        5.99912643432617188e-01, 6.07329845428466797e-01,
        6.14709377288818359e-01, 6.22051715850830078e-01,
        6.29356384277343750e-01, 6.36624336242675781e-01,
        6.43856048583984375e-01, 6.51051521301269531e-01,
        6.58211231231689453e-01, 6.65335655212402344e-01,
        6.72425270080566406e-01, 6.79480075836181641e-01,
        6.86500072479248047e-01, 6.93486690521240234e-01,
        7.00439453125000000e-01, 7.07358837127685547e-01,
        7.14245319366455078e-01, 7.21098899841308594e-01,
        7.27920055389404297e-01, 7.34709262847900391e-01,
        7.41466522216796875e-01, 7.48192787170410156e-01,
        7.54887104034423828e-01, 7.61550903320312500e-01,
        7.68184185028076172e-01, 7.74786949157714844e-01,
        7.81359672546386719e-01, 7.87902355194091797e-01,
        7.94415473937988281e-01, 8.00899505615234375e-01,
        8.07354450225830078e-01, 8.13780784606933594e-01,
        8.20178508758544922e-01, 8.26548099517822266e-01,
        8.32889556884765625e-01, 8.39203357696533203e-01,
        8.45489978790283203e-01, 8.51748943328857422e-01,
        8.57980728149414062e-01, 8.64185810089111328e-01,
        8.70364665985107422e-01, 8.76516819000244141e-01,
        8.82642745971679688e-01, 8.88742923736572266e-01,
        8.94817352294921875e-01, 9.00866508483886719e-01,
        9.06890392303466797e-01, 9.12889003753662109e-01,
        9.18862819671630859e-01, 9.24812316894531250e-01,
        9.30737018585205078e-01, 9.36637878417968750e-01,
        9.42514419555664062e-01, 9.48367118835449219e-01,
        9.54195976257324219e-01, 9.60001468658447266e-01,
        9.65784072875976562e-01, 9.71543312072753906e-01,
        9.77279663085937500e-01, 9.82993125915527344e-01,
  0.00000000000000000e+00, 1.12272500991821289e-02, 2.23678052425384521e-02,
  3.34229767322540283e-02, 4.43941056728363037e-02, 5.52824139595031738e-02,
  6.60891532897949219e-02, 7.68155455589294434e-02, 8.74627828598022461e-02,
  9.80320572853088379e-02, 1.08524441719055176e-01, 1.18941068649291992e-01,
  1.29282951354980469e-01, 1.39551281929016113e-01, 1.49747014045715332e-01,
  1.59871220588684082e-01, 1.69924974441528320e-01, 1.79908990859985352e-01,
  1.89824461936950684e-01, 1.99672341346740723e-01, 2.09453344345092773e-01,
  2.19168424606323242e-01, 2.28818655014038086e-01, 2.38404631614685059e-01,
  2.47927427291870117e-01, 2.57387638092041016e-01, 2.66786336898803711e-01,
  2.76124238967895508e-01, 2.85402059555053711e-01, 2.94620513916015625e-01,
  3.03780555725097656e-01, 3.12882900238037109e-01, 3.21928024291992188e-01,
  3.30916643142700195e-01, 3.39849948883056641e-01, 3.48727941513061523e-01,
  3.57551813125610352e-01, 3.66322040557861328e-01, 3.75039339065551758e-01,
  3.83704185485839844e-01, 3.92317295074462891e-01, 4.00879383087158203e-01,
  4.09309263610839984e-01, 4.17852401733398438e-01, 4.26264524459838867e-01,
  4.34628009796142578e-01, 4.42943334579467773e-01, 4.51210975646972656e-01,
  4.59431409835815430e-01, 4.67605352401733398e-01, 4.75733280181884766e-01,
  4.83815670013427734e-01, 4.91852998733520508e-01, 4.99845743179321289e-01,
  5.07794380187988281e-01, 5.15699386596679688e-01, 5.23561954498291016e-01,
  5.31381130218505859e-01, 5.39158344268798828e-01, 5.46894073486328125e-01,
  5.54588794708251953e-01, 5.62242031097412109e-01, 5.69855213165283203e-01,
  5.77428817749023438e-01, 5.84962368011474609e-01, 5.92456817626953125e-01,
  5.99912643432617188e-01, 6.07329845428466797e-01, 6.14709377288818359e-01,
  6.22051715850830078e-01, 6.29356384277343750e-01, 6.36624336242675781e-01,
  6.43856048583984375e-01, 6.51051521301269531e-01, 6.58211231231689453e-01,
  6.65335655212402344e-01, 6.72425270080566406e-01, 6.79480075836181641e-01,
  6.86500072479248047e-01, 6.93486690521240234e-01, 7.00439453125000000e-01,
  7.07358837127685547e-01, 7.14245319366455078e-01, 7.21098899841308594e-01,
  7.27920055389404297e-01, 7.34709262847900391e-01, 7.41466522216796875e-01,
  7.48192787170410156e-01, 7.54887104034423828e-01, 7.61550903320312500e-01,
```

```
 7.68184185028076172e-01, 7.74786949157714844e-01, 7.81359672546386719e-01,
 7.87902355194091797e-01, 7.94415473937988281e-01, 8.00899505615234375e-01,
 8.07354450225830078e-01, 8.13780784606933594e-01, 8.20178508758544922e-01,
 8.26548099517822266e-01, 8.32889556884765625e-01, 8.39203357696533203e-01,
 8.45489978790283203e-01, 8.51748943328857422e-01, 8.57980728149414062e-01,
 8.64185810089111328e-01, 8.70364665985107422e-01, 8.76516819000244141e-01,
 8.82642745971679688e-01, 8.88742923736572266e-01, 8.94817352294921875e-01,
 9.00866508483886719e-01, 9.06890392303466797e-01, 9.12889003753662109e-01,
 9.18862819671630859e-01, 9.24812316894531250e-01, 9.30737018585205078e-01,
 9.36637878417968750e-01, 9.42514419555664062e-01, 9.48367118835449219e-01,
 9.54195976257324219e-01, 9.60001468658447266e-01, 9.65784072875976562e-01,
 9.71543312072753906e-01, 9.77279663085937500e-01, 9.82993125915527344e-01,
        9.88684654235839844e-01, 9.94353294372558594e-01,
};

#endif /* ! codereview */
const double _TBL_log2_lo[] = {
        0.00000000000000000e+00, 5.32407199143163062e-09,
        7.78591605611869461e-09, 2.48051962506972834e-08,
        1.36856171339421649e-08, 2.15416864274073636e-08,
        3.71679775110542797e-08, 5.14919014488721604e-08,
        5.83905371621603131e-08, 2.56752178779050280e-08,
        1.50591138779666358e-08, 4.07421543880223335e-09,
        6.55899859865622946e-08, 7.04697774403433060e-08,
        1.05458966729375492e-07, 1.16189705334564924e-07,
        2.70007840425949794e-08, 9.91549491170275978e-08,
        9.69430665462702729e-08, 3.48962367368142750e-09,
        2.12838570084203029e-08, 9.58558383294243244e-08,
        3.54818427912568755e-08, 1.07710393847949145e-07,
        8.61517153766060168e-08, 2.04600610755536536e-07,
        2.03796097652703831e-07, 1.66306342048863931e-07,
        1.59307194630913047e-07, 2.34975611381410033e-07,
        1.92452005268177275e-07, 5.50463182513595194e-08,
        7.05953701603703195e-08, 2.34971916784423615e-07,
        5.40015680851899589e-08, 2.12718016029126278e-07,
        1.91492473341603465e-07, 1.73687954457398432e-07,
        9.22813729985471341e-08, 1.06988212380721318e-07,
        1.27704297398270718e-07, 5.31950261176686284e-08,
        9.77661777174938596e-09, 1.13152499419201003e-07,
        2.30242259071696645e-07, 2.17840582054596399e-07,
        1.61269260528736021e-07, 1.36185356146932601e-07,
        2.08801481826511869e-07, 1.97681264041823641e-07,
        1.55872185666914818e-09, 3.30297806270353139e-07,
        4.66839232562134881e-07, 3.86401308539453419e-07,
        5.69693854190458130e-08, 3.93123660542428204e-07,
        3.95165664638538863e-07, 1.02867252517587785e-08,
        1.32709681572078730e-07, 2.19641127294637299e-07,
        1.98754510492326232e-07, 4.68321143892845854e-07,
        4.66826389855508924e-07, 1.03605546188658804e-07,
        2.35802265869106829e-07, 2.84300973057307715e-07,
        1.41190740320740639e-07, 1.69877659083133016e-07,
        2.51520105284046651e-07, 2.61972773884411727e-07,
        7.18909291834578061e-08, 2.36692644004112907e-08,
        4.54703970334185855e-07, 2.66978085000826612e-07,
        2.65016092160396791e-07, 2.94953197203117899e-07,
        1.98299667558641024e-07, 2.88865876540408914e-07,
        3.99173794882405776e-07, 3.57377937852235498e-07,
        4.64184350072864601e-07, 6.24190501305044646e-08,
        3.98129044716236242e-07, 3.29124166816248113e-07,
        1.39748850186603795e-07, 1.10443458567567753e-07,
        4.09782728853196823e-08, 2.04197339771775867e-07,
        3.92412117682061536e-07, 3.94305070358032831e-07,
        4.71831774029316962e-07, 4.06610103464898125e-07,
```



```
        4.53656642786443564e-07, 3.87773092718157073e-07,
        4.57279976050247260e-07, 4.30400410735578705e-07,
        7.21540920170394723e-08, 9.80872001232200742e-08,
        2.66978158058219765e-07, 3.34565168908893463e-07,
        5.35982971014292903e-08, 1.27564755579416119e-07,
        3.03390161571307385e-07, 3.25161686840256005e-07,
        4.11013021640696012e-07, 2.99496861839592342e-07,
        2.03305051732449063e-07, 3.32476299509608735e-07,
        4.17602963653023739e-07, 1.86711249657268702e-07,
        3.18977681198347184e-07, 6.05846018127542565e-08,
        8.57835758121197076e-08, 1.12749228435440334e-07,
        3.34129550990056099e-07, 4.63409633672188390e-07,
        2.11786110481110945e-07, 2.41878018084726962e-07,
        2.60413978970349421e-07, 4.48778782784743522e-07,
 0.00000000000000000e+00, 5.32407199143163062e-09, 7.78591605611869461e-09,
 2.48051962506972834e-08, 1.36856171339421649e-08, 2.15416864274073636e-08,
 3.71679775110542797e-08, 5.14919014488721604e-08, 5.83905371621603131e-08,
 2.56752178779050280e-08, 1.50591138779666358e-08, 4.07421543880223335e-09,
 6.55899859865622946e-08, 7.04697774403433060e-08, 1.05458966729375492e-07,
 1.16189705334564924e-07, 2.70007840425949794e-08, 9.91549491170275978e-08,
 9.69430665462702729e-08, 3.48962367368142750e-09, 2.12838570084203029e-08,
 9.58558383294243244e-08, 3.54818427912568755e-08, 1.07710393847949145e-07,
 8.61517153766060168e-08, 2.04600610755536536e-07, 2.03796097652703831e-07,
 1.66306342048863931e-07, 1.59307194630913047e-07, 2.34975611381410033e-07,
 1.92452005268177275e-07, 5.50463182513595194e-08, 7.05953701603703195e-08,
 2.34971916784423615e-07, 5.40015680851899589e-08, 2.12718016029126278e-07,
 1.91492473341603465e-07, 1.73687954457398432e-07, 9.22813729985471341e-08,
 1.06988212380721318e-07, 1.27704297398270718e-07, 5.31950261176686284e-08,
 9.77661777174938596e-09, 1.13152499419201003e-07, 2.30242259071696645e-07,
 2.17840582054596399e-07, 1.61269260528736021e-07, 1.36185356146932601e-07,
 2.08801481826511869e-07, 1.97681264041823641e-07, 1.55872185666914818e-09,
 3.30297806270353139e-07, 4.66839232562134881e-07, 3.86401308539453419e-07,
 5.69693854190458130e-08, 3.93123660542428204e-07, 3.95165664638538863e-07,
 1.02867252517587785e-08, 1.32709681572078730e-07, 2.19641127294637299e-07,
 1.98754510492326232e-07, 4.68321143892845854e-07, 4.66826389855508924e-07,
 1.03605546188658804e-07, 2.35802265869106829e-07, 2.84300973057307715e-07,
 1.41190740320740639e-07, 1.69877659083133016e-07, 2.51520105284046651e-07,
 2.61972773884411727e-07, 7.18909291834578061e-08, 2.36692644004112907e-08,
 4.54703970334185855e-07, 2.66978085000826612e-07, 2.65016092160396791e-07,
 2.94953197203117899e-07, 1.98299667558641024e-07, 2.88865876540408914e-07,
 3.99173794882405776e-07, 3.57377937852235498e-07, 4.64184350072864601e-07,
 6.24190501305044646e-08, 3.98129044716236242e-07, 3.29124166816248113e-07,
 1.39748850186603795e-07, 1.10443458567567753e-07, 4.09782728853196823e-08,
 2.04197339771775867e-07, 3.94305070358032831e-07, 4.71831774029316962e-07,
 4.06610103464898125e-07,
        3.25363260095300064e-08, 1.42486299343828112e-07,
};
```

_____*unchanged_portion_omitted_*

```
**********************************************************
   44767 Thu Jun  6 20:37:31 2019
new/usr/src/lib/libm/common/C/_TBL_sin.c
11210 libm should be cstyle(10NBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

#endif /* ! codereview */
/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2005 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#include "libm_protos.h"

/*
 * Table of constants for x[i],sin(x[i]),cos(x[i]), where
 * x[i] ~ (i+10.5)/64 chosen to make the value of sine and
 * cosine nearly representable in double (with error less
 * than 2**-8 ulp)
 * By K.C. Ng, May 5, 1995
 *
 * For each i, _TBL_sincosx[i] := x[i], _TBL_sincos[2*i] :=
 * sin(x[i]), and _TBL_sincos[2*i+1] := cos(x[i]).
 */

const double _TBL_sincos[] = {
        1.63327491736778435127e-01, 9.86571908399470176576e-01,
        1.78722113534634630128e-01, 9.83899591489758251761e-01,
        1.94073102892906523831e-01, 9.80987069605669836925e-01,
        2.09376712086097482857e-01, 9.77835053797937558961e-01,
        2.24629204957583178404e-01, 9.74444313586017130113e-01,
        2.39826857830661321902e-01, 9.70815676770349522684e-01,
        2.54965960415442560727e-01, 9.66950029230792762469e-01,
        2.70042816718758793559e-01, 9.62848314709330965755e-01,
        2.85053745940880454146e-01, 9.58511534581129587274e-01,
        2.99995083378835347698e-01, 9.53940747608846839611e-01,
        3.14863181320744367486e-01, 9.49137069684131584602e-01,
        3.29654409930721814526e-01, 9.44101673557052656349e-01,
        3.44365158144533722862e-01, 9.38835788546692695533e-01,
        3.58991834544317267586e-01, 9.33340700243220688925e-01,
        3.73530868238515501023e-01, 9.27617750192923362640e-01,
```



```
        3.87978709726743087316e-01, 9.21668335573470609567e-01,
        4.02331831777567594521e-01, 9.15493908848391546584e-01,
        4.16586730281922223984e-01, 9.09095977415485534401e-01,
        4.30739925110786514573e-01, 9.02476103237949467406e-01,
        4.44787960958008266044e-01, 8.95635902466408118094e-01,
        4.58727408216676513231e-01, 8.88577045028066558885e-01,
        4.72554863751536879946e-01, 8.81301254251215970825e-01,
        4.86266951795427115890e-01, 8.73810306411857196096e-01,
        4.99860324731856597857e-01, 8.66106030321324382726e-01,
        5.13331663943585647658e-01, 8.58190306862591900661e-01,
        5.26677680590333596733e-01, 8.50065068549453184410e-01,
        5.39895116435048061376e-01, 8.41732299041438647436e-01,
        5.52980744632255882820e-01, 8.33194032663434169805e-01,
        5.65931370507619768695e-01, 8.24452353914625679643e-01,
        5.78743832357296650315e-01, 8.15509396946711651033e-01,
        5.91415002201596706755e-01, 8.06367345054898265744e-01,
        6.03941786558566895415e-01, 7.97028430138126520177e-01,
        6.16321127179607297641e-01, 7.87494932169127248578e-01,
        6.28550001844884853597e-01, 7.77769178600434929471e-01,
        6.40625425044079821468e-01, 7.67853543839638774671e-01,
        6.52544448725672743272e-01, 7.57750448655299613243e-01,
        6.64304163044103668234e-01, 7.47462359562187539375e-01,
        6.75901697026429104653e-01, 7.36991788256011193248e-01,
        6.87334219302880855551e-01, 7.26341290975047959577e-01,
        6.98598938789923074033e-01, 7.15513467882745946014e-01,
        7.09693105361432152733e-01, 7.04510962443060329008e-01,
        7.20614010544995853280e-01, 6.93336460750663685637e-01,
        7.31358988151144640000e-01, 6.81992690906972898190e-01,
        7.41925414945620254059e-01, 6.70482422333180339002e-01,
        7.52310711296420575600e-01, 6.58808465085774175307e-01,
        7.62512341773335489137e-01, 6.46973669204044199432e-01,
        7.72527815799416095466e-01, 6.34980923978180178402e-01,
        7.82354688238184881044e-01, 6.22833157267443926486e-01,
        7.91990560000511156780e-01, 6.10533347738489679 91e-01,
        8.01433078627164507957e-01, 5.98084459321745920413e-01,
        8.10679938859144910701e-01, 5.85489570130274028514e-01,
        8.19728883213368231253e-01, 5.72751742053888568407e-01,
        8.28577702516849257108e-01, 5.59874084854710574177e-01,
        8.37224236455711978699e-01, 5.46859742430497508536e-01,
        8.45666374107491569667e-01, 5.33711892039036461810e-01,
        8.53902054441761149128e-01, 5.20433743544881588505e-01,
        8.61929266833302509809e-01, 5.07028538621057900393e-01,
        8.69746051561515076678e-01, 4.93499549942200410602e-01,
        8.77350500260862697921e-01, 4.79850080433476600117e-01,
        8.84740756420631879742e-01, 4.66083462405874393575e-01,
        8.91915015812867362222e-01, 4.52203056787028545571e-01,
        8.98871526946913745881e-01, 4.38212252275223035358e-01,
        9.05608591487805036913e-01, 4.24114464529888268718e-01,
        9.12124564678846838639e-01, 4.09913135321892496687e-01,
        9.18417855741508804002e-01, 3.95611731695571844369e-01,
        9.24486928255549345046e-01, 3.81213745141251114656e-01,
        9.30330300545781363475e-01, 3.66722690716563270996e-01,
        9.35946546034209125864e-01, 3.52142106210879102246e-01,
        9.41334293596668869597e-01, 3.37475551260917883134e-01,
        9.46492227896101323559e-01, 3.22726606483374089951e-01,
        9.51419089686698082886e-01, 3.07898872650964328113e-01,
        9.56113676155394554002e-01, 2.92995969713948978264e-01,
        9.60574841181938254842e-01, 2.78021536015277237475e-01,
        9.64801495637480077683e-01, 2.62979227346346711158e-01,
        9.68792607644664016675e-01, 2.47872716072285947941e-01,
        9.72547202831614887586e-01, 2.32705690227810568782e-01,
        9.76064364566613607010e-01, 2.17481852629530458820e-01,
        9.79343234187565414572e-01, 2.02204919947659544910e-01,
        9.82383011202836109454e-01, 1.86878621837941599759e-01,
        9.85182953494231017366e-01, 1.71506699998524386741e-01,
        9.87742377497998091940e-01, 1.56092907252707135957e-01,
```

```
        9.90060658366647028394e-01, 1.40641006660935985462e-01,
        9.92137230124395808062e-01, 1.25154770588626285122e-01,
        9.93971585806359803072e-01, 1.09637979777038541140e-01,
        9.95563277581850036846e-01, 9.40944224196323536491e-02,
        9.96911916861350277941e-01, 7.85278932598362233719e-02,
        9.98017174394052908326e-01, 6.29421926414276133865e-02,
        9.98878780347215333713e-01, 4.73411255827534852866e-02,
        9.99496524372108563483e-01, 3.17285008797294557081e-02,
        9.99870255655346151791e-01, 1.61081301122361006395e-02,
        9.99999882955821872699e-01, 4.83826769160181427432e-04,
        9.99885374626887313276e-01, -1.51405946795101862407e-02,
        9.99526758624139421983e-01, -3.07613197753498594789e-02,
        9.98924122498464628350e-01, -4.63745349375326090802e-02,
        9.98077613374894423437e-01, -6.19764284214149308028e-02,
        9.96987437916807328619e-01, -7.75631912448182664344e-02,
        9.95653862273598311283e-01, -9.31310181393209396417e-02,
        9.94077212020575529117e-01, -1.08676108420387079745e-01,
        9.92257872072439317535e-01, -1.24194666996109981394e-01,
        9.90196286596708996619e-01, -1.39682905217811598186e-01,
        9.87892958898728967831e-01, -1.55137041864005509328e-01,
        9.85348451302295424981e-01, -1.70553304031812708041e-01,
        9.82563385014030843401e-01, -1.85927928052160545969e-01,
        9.79538439968065888230e-01, -2.01257160431443482551e-01,
        9.76274354660002341433e-01, -2.16537258764389006771e-01,
        9.72771925969731610095e-01, -2.31764492632368090952e-01,
        9.69032008956924317822e-01, -2.46935144556029828600e-01,
        9.65055516693764658953e-01, -2.62045510739892129060e-01,
        9.60843419958733790942e-01, -2.77091902303196746526e-01,
        9.56396747083171572257e-01, -2.92070645852553878452e-01,
        9.51716583658057113659e-01, -3.06978084543891138747e-01,
        9.46804072278775166183e-01, -3.21810578937871294425e-01,
        9.41660412264228252610e-01, -3.36564507894643705210e-01,
        9.36286859366077139910e-01, -3.51236269451786098372e-01,
        9.30684725460523609719e-01, -3.65822281708577445869e-01,
        9.24855378224429758305e-01, -3.80318983708869018390e-01,
        9.18800240811794344253e-01, -3.94722836284130795814e-01,
        9.12520791499566663596e-01, -4.09030322935848345001e-01,
        9.06018563323250702979e-01, -4.23237950701107090712e-01,
        8.99295143708603639254e-01, -4.37342250991282488481e-01,
        8.92352174084417359978e-01, -4.51339780439098559039e-01,
        8.85191349474114597129e-01, -4.65227121754735961634e-01,
        8.77814418087698666859e-01, -4.79000884547570504601e-01,
        8.70223180928641018 60e-01, -4.92657706140177065190e-01,
        8.62419491209962973954e-01, -5.06194252418129098103e-01,
        8.54405254167239447405e-01, -5.19607218629047684644e-01,
        8.46182426332270809510e-01, -5.32893330195106762481e-01,
        8.37753015193838712626e-01, -5.46049343497116423940e-01,
        8.29119078677651999421e-01, -5.59072046674417011403e-01,
        8.20282724626069215113e-01, -5.71958260435175724901e-01,
        8.11246110312714763246e-01, -5.84704838788333125521e-01,
        8.02011441899084687179e-01, -5.97308669837422590021e-01,
        7.92580973890125495274e-01, -6.09766676547169317324e-01,
        7.82957008603788473522e-01, -6.22075817467780289860e-01,
        7.73141895594474215514e-01, -6.34233087497477643346e-01,
        7.63138031079152456826e-01, -6.46235518615801973752e-01,
        7.52947857359473227135e-01, -6.58080180599429964694e-01,
        7.42573862219235825144e-01, -6.69764181745192588302e-01,
        7.32018578314804879703e-01, -6.81284669577975954269e-01,
        7.21284582577006005977e-01, -6.92638831525286491342e-01,
        7.10374495555637031075e-01, -7.03823895633044149811e-01,
        6.99290980797484418297e-01, -7.14837131223114541356e-01,
        6.88036744157449198234e-01, -7.25675849597612554476e-01,
        6.76614533221899572268e-01, -7.36337404613476742554e-01,
        6.65027136549188546688e-01, -7.46819193415104276568e-01,
        6.53277383052505156158e-01, -7.57118657009633100330e-01,
        6.41368141233487065733e-01, -7.67233280958732777322e-01,
```



```
        6.29302318589868403542e-01, -7.77160595898567008177e-01,
        6.17082860810903133242e-01, -7.86898178224750721732e-01,
        6.04712751105658807838e-01, -7.96443650643424705393e-01,
        5.92195009450509846083e-01, -8.05794682770934245220e-01,
        5.79532691867931770702e-01, -8.14948991689853463605e-01,
        5.66728889706594629594e-01, -8.23904342488817387213e-01,
        5.53786728799491090314e-01, -8.32658548869558479133e-01,
        5.40709368819720759269e-01, -8.41209473597735457595e-01,
        5.27500002380493770993e-01, -8.49555029111463189118e-01,
        5.14161854409658891640e-01, -8.57693177931374672873e-01,
        5.00698181184736190730e-01, -8.65621933270118271153e-01,
        4.87112269682015319727e-01, -8.73339359427499739574e-01,
        4.73407436683839610847e-01, -8.80843572317148937323e-01,
        4.59587028080454429446e-01, -8.88132739865035936155e-01,
        4.45654417892204612883e-01, -8.95205082544307417791e-01,
        4.31613007576607476956e-01, -9.02058873738668665077e-01,
        4.17466225094956511210e-01, -9.08692440215591923369e-01,
        4.03217524247773739798e-01, -9.15104162453376668296e-01,
        3.88870383625079307777e-01, -9.21292475134407928827e-01,
        3.74428305866518040812e-01, -9.27255867474522488259e-01,
        3.59894816812003803808e-01, -9.32992883591217014860e-01,
        3.45273464602750546071e-01, -9.38502122875176647554e-01,
        3.30567818825136694461e-01, -9.43782240327286303661e-01,
        3.15781469657649860316e-01, -9.48831946880402399280e-01,
        3.00918026974915431282e-01, -9.53650009721346392233e-01,
        2.85981119468962208252e-01, -9.58235252590552089025e-01,
        2.70974393771316324209e-01, -9.62586556066657106356e-01,
        2.55901513568614069616e-01, -9.66702857838587559236e-01,
        2.40766158683884484715e-01, -9.70583152971761897732e-01,
        2.25572024178931879179e-01, -9.74226494152062860721e-01,
        2.10322819513115238932e-01, -9.77631991902911057224e-01,
        1.95022267545207572681e-01, -9.80798814824694553671e-01,
        1.79674103687683967001e-01, -9.83726189782516358129e-01,
        1.64282074965636487596e-01, -9.86413402101261493904e-01,
        1.48849939140241666058e-01, -9.88859795733422641817e-01,
        1.33381463740289751829e-01, -9.91064773428305123559e-01,
        1.17880425165185737102e-01, -9.93027796873216961338e-01,
        1.02350607771443738447e-01, -9.94748386823932517764e-01,
        8.67958029390951818494e-02, -9.96226123223115322958e-01,
        7.12198081674702832000e-02, -9.97460453011510831 53e-01,
        5.56264261071372570489e-02, -9.98451651667994988237e-01,
        4.00194636390110436430e-02, -9.99198900384726140800e-01,
        2.44027309972172715136e-02, -9.99702209020204901613e-01,
        8.78004077991816241078e-03, -9.99961454699081375708e-01,
        -6.84479296391702837776e-03, -9.99976574130254869388e-01,
        -2.24679556394218951643e-02, -9.99747563622630175395e-01,
        -3.80856331006515710924e-02, -9.99274479085362488107e-01,
        -5.36940124898220294547e-02, -9.98557436015947375019e-01,
        -6.92898232575160572128e-02, -9.97596609469809547655e-01,
        -8.48676380386628043118e-02, -9.96392234019183087312e-01,
        -1.00425273601341916163e-01, -9.94944603695148255262e-01,
        -1.15958391781735684067e-01, -9.93254071914831615508e-01,
        -1.31463200384306394541e-01, -9.91321051397939245753e-01,
        -1.46935914119801724897e-01, -9.89146014065556578032e-01,
        -1.62372755568482129984e-01, -9.86729490918913376696e-01,
        -1.77769956039573850948e-01, -9.84072071918356994225e-01,
        -1.93123756521520834051e-01, -9.81174405835688490107e-01,
        -2.08430408606563005725e-01, -9.78037200094199477007e-01,
        -2.23686175400125447643e-01, -9.74661220596605204491e-01,
        -2.38887332428331961021e-01, -9.71047291539024692852e-01,
        -2.54030168529570332669e-01, -9.67196295214595047618e-01,
        -2.69110986809851404633e-01, -9.63109171785954898404e-01,
        -2.84126105504238113397e-01, -9.58786919065437892584e-01,
        -2.99071858881536201125e-01, -9.54230592706222354 18e-01,
        -3.13944598143160502612e-01, -9.49441303765919730751e-01,
        -3.28740692363219233485e-01, -9.44420222774031481450e-01,
```

```
        -3.43456529243486463621e-01, -9.39168575134420757777e-01,
        -3.58088516132365641820e-01, -9.33687642958886176991e-01,
        -3.72633080853157161449e-01, -9.27978764333475703019e-01,
        -3.87086672547184373894e-01, -9.22043333003578990947e-01,
        -4.01445762590873223008e-01, -9.15882798013933796533e-01,
        -4.15706845395529489551e-01, -9.09498663380709615467e-01,
        -4.29866439353555507275e-01, -9.02892487684716527063e-01,
        -4.43921087571260808424e-01, -8.96065883743795366101e-01,
        -4.57867358817895864220e-01, -8.89020518171051099543e-01,
        -4.71701848327647499381e-01, -8.81758110982984399939e-01,
        -4.85421178579811707365e-01, -8.74280435207254624785e-01,
        -4.99022000232008211551e-01, -8.66589316391822128693e-01,
        -5.12500992809901023683e-01, -8.58686632229048840692e-01,
        -5.25854865641323332426e-01, -8.50574312027670975667e-01,
        -5.39080358520030999969e-01, -8.42254336324791408330e-01,
        -5.52174242663304060130e-01, -8.33728736304085060738e-01,
        -5.65133321393192722404e-01, -8.24999593364201810886e-01,
        -5.77954430931352902689e-01, -8.16069038603239760299e-01,
        -5.90634441175508673183e-01, -8.06939252296785092256e-01,
        -6.03170256463835929850e-01, -7.97612463366358381833e-01,
        -6.15558816459891189332e-01, -7.88090948735295393490e-01,
        -6.27797096543907584554e-01, -7.78377033044423516372e-01,
        -6.39882108993420795073e-01, -7.68473087746169514212e-01,
        -6.51810903392718188343e-01, -7.58381530773507561705e-01,
        -6.63580567511655061708e-01, -7.48104825823834529430e-01,
        -6.75188227925781481176e-01, -7.37645481834222960238e-01,
        -6.86631050850229573967e-01, -7.27006052250123602221e-01,
        -6.97906242654146802273e-01, -7.16189134561793783185e-01,
        -7.09011050643817641870e-01, -7.05197369581700761465e-01,
        -7.19942763756367454242e-01, -6.94033440775618015728e-01,
        -7.30698713155769064009e-01, -6.82700073672548479742e-01,
        -7.41276272975477157345e-01, -6.71200035103981407225e-01,
        -7.51672860805046583188e-01, -6.59536132694151233657e-01,
        -7.61885938516202787518e-01, -6.47711213961349341339e-01,
        -7.71913012640803364306e-01, -6.35728165897814334606e-01,
        -7.81751635309322678857e-01, -6.23589913878664137137e-01,
        -7.91399404523052685256e-01, -6.11299421331770620469e-01,
        -8.00853964899717496451e-01, -5.98859688829029512824e-01,
        -8.10113008319712335492e-01, -5.86273753251146056975e-01,
        -8.19174274236826760465e-01, -5.73544687385881157837e-01,
        -8.28035550507897455397e-01, -5.60675598804766250893e-01,
        -8.36694673776658404130e-01, -5.47669629314764150330e-01,
        -8.45149530028187490061e-01, -5.34529954158916909002e-01,
        -8.53398055161871504914e-01, -5.21259781151332757254e-01,
        -8.61438235389631601358e-01, -5.07862350060326539491e-01,
        -8.69268107829002323328e-01, -4.94340931656873816546e-01,
        -8.76885760925650292741e-01, -4.80698827006935058836e-01,
        -8.84289334936661730602e-01, -4.66939366639049280305e-01,
        -8.91477022398163843064e-01, -4.53065909704210345588e-01,
        -8.98447068525225711610e-01, -4.39081843234753188554e-01,
        -9.05197771673453388530e-01, -4.24990581257296273776e-01,
        -9.11727483791179293959e-01, -4.10795563875518132679e-01,
        -9.18034610707084031134e-01, -3.96500256675695827990e-01,
        -9.24117612643078456536e-01, -3.82108149615860981374e-01,
        -9.29975004511545022545e-01, -3.67622756346437040698e-01,
        -9.35605356329172521690e-01, -3.53047613231079804308e-01,
        -9.41007293511755382731e-01, -3.38386278619096869669e-01,
        -9.46179497257704227309e-01, -3.23642331855951870256e-01,
        -9.51120704853153031699e-01, -3.08819372448752571536e-01,
        -9.55829709968717189383e-01, -2.93921019223052470970e-01,
        -9.60305362967905695726e-01, -2.78950909399985458315e-01,
        -9.64546571183209522360e-01, -2.63912697721639999404e-01,
        -9.68552299193694232748e-01, -2.48810055517474232323e-01,
        -9.72321569045517364316e-01, -2.33646669928897571245e-01,
        -9.75853460530087812863e-01, -2.18426242863471758993e-01,
        -9.79147111396304836717e-01, -2.03152490125698942380e-01,
```



```
        -9.82201717531947959827e-01, -1.87829140649930864670e-01,
        -9.85016533205280153673e-01, -1.72459935382833828843e-01,
        -9.87590871221861066331e-01, -1.57048626479970948600e-01,
        -9.89924103089018792012e-01, -1.41589876420741456961e-01,
        -9.92015659185421450061e-01, -1.26114756991058563074e-01,
        -9.93865028889118318212e-01, -1.10599748423005059261e-01,
        -9.95471760691319929037e-01, -9.50577385914659067634e-02,
        -9.96835462344218936614e-01, -7.94925217425341834598e-02,
        -9.97955800916290658442e-01, -6.39078979276023889655e-02,
        -9.98832502892746831868e-01, -4.83076719063431220258e-02,
        -9.99465354238023406808e-01, -3.26956522776712110723e-02,
        -9.99854200451614993916e-01, -1.70756504784357332483e-02,
        -9.99998946602528415717e-01, -1.45147987706449187358e-03,
        -9.99899557352339485305e-01, 1.41730450713867285606e-02,
        -9.99556056965451689145e-01, 2.97941098823021957576e-02,
        -9.98968529303411734155e-01, 4.54079008695470534573e-02,
        -9.98137117802025963798e-01, 6.10106061751933687054e-02,
        -9.97062025438244736719e-01, 7.65984166219185330649e-02,
        -9.95743514682696617690e-01, 9.21675266422526118237e-02,
        -9.94181907425219280050e-01, 1.07714135322866152999e-01,
        -9.92377584917212285376e-01, 1.23234447107459038628e-01,
        -9.90330987653228356216e-01, 1.38724672981345553691e-01,
        -9.88042615281836456020e-01, 1.54181031216647779214e-01,
        -9.85513026476385278762e-01, 1.69599748364658631239e-01,
        -9.82742838804682716791e-01, 1.84977060140206039929e-01,
        -9.79732728555263054915e-01, 2.00309212463279484595e-01,
        -9.76483430616286174342e-01, 2.15592462140605983789e-01,
        -9.72995738247511954278e-01, 2.30823078032026951512e-01,
        -9.69270502929450938900e-01, 2.45997341755737758406e-01,
        -9.65308634114379171542e-01, 2.61111548776057855736e-01,
        -9.61111099038787108917e-01, 2.76162009162112587202e-01,
        -9.56678922485658334018e-01, 2.91145048509638459944e-01,
        -9.52013186489632401432e-01, 3.06057008986653333871e-01,
        -9.47115030121562395671e-01, 3.20894250054175877995e-01,
        -9.41985649202698782645e-01, 3.35653149391108962529e-01,
        -9.36626296000886870985e-01, 3.50330103815899684960e-01,
        -9.31038278925287121623e-01, 3.64921530162087393023e-01,
        -9.25222962204842236389e-01, 3.79423866156172462372e-01,
        -9.19181765559584973424e-01, 3.93833571274420646269e-01,
        -9.1291613872961705650e-01, 4.08147127564896294860e-01,
        -9.06427686803489396361e-01, 4.22361040575566504263e-01,
        -8.99717918410242289973e-01, 4.36471840204543548580e-01,
        -8.92788496793018526709e-01, 4.50476081489419755144e-01,
        -8.85641113671704172106e-01, 4.64370345494136360642e-01,
        -8.78277513965914136129e-01, 4.78151240155093082418e-01,
        -8.70699495405757306621e-01, 4.91815401040023969514e-01,
        -8.62908908048144129843e-01, 5.05359492253939501794e-01,
        -8.54907653871092576559e-01, 5.18780207171229856833e-01,
        -8.46697686222891654495e-01, 5.32074269388026044325e-01,
        -8.38281009508205721126e-01, 5.45238433254574883513e-01,
        -8.29659678498936070667e-01, 5.58269484991829045839e-01,
        -8.20835797971514846694e-01, 5.71164243250981584765e-01,
        -8.11811522157973475267e-01, 5.83919559949445554636e-01,
        -8.02589054191142126093e-01, 5.96532321079560556853e-01,
        -7.93170645644559635379e-01, 6.08999447362468804279e-01,
        -7.83558595847759331576e-01, 6.21317895181756174594e-01,
        -7.73755251444074421130e-01, 6.33484657164415598807e-01,
        -7.63763005819449558587e-01, 6.45496762921116018497e-01,
        -7.53584298396099971917e-01, 6.57351279918779618505e-01,
        -7.43221614171602262822e-01, 6.69045314031985416392e-01,
        -7.32677483058112311021e-01, 6.80576010317458623966e-01,
        -7.21954479231692536345e-01, 6.91940553745258979390e-01,
        -7.11055220593523329420e-01, 7.03136169789818077369e-01,
        -6.99982367997418419847e-01, 7.14160125246941168697e-01,
        -6.88738624756222161949e-01, 7.25009728740868553132e-01,
        -6.77326735865867002317e-01, 7.35682331500009611958e-01,
```

```
        -6.6574948736052919073 8e-01, 7.4617532797539787292 6e-01,
        -6.5400970566742766543 2e-01, 7.5648615644491778997 6e-01,
        -6.4211025687859724087 0e-01, 7.6661229967389765693 8e-01,
        -6.3005404606977988279 9e-01, 7.7655128551248930879 3e-01,
        -6.1784401664170951473 7e-01, 7.8630068745998116241 9e-01,
        -6.0548314942781145120 4e-01, 7.9585812539609002147 5e-01,
        -5.9297446218407845464 1e-01, 8.0522126598687326914 9e-01,
        -5.8032100874022618519 6e-01, 8.1438782334630122061 7e-01,
        -5.6752587824818723216 7e-01, 8.2335555959659600944 2e-01,
        -5.5459219446065222136 6e-01, 8.3212228539038546326 6e-01,
        -5.4152311492198534903 5e-01, 8.4068586047654580983 8e-01,
        -5.2832183027922297036 1e-01, 8.4904419416801379938 4e-01,
        -5.1499156344548402408 6e-01, 8.5719524589207563014 5e-01,
        -5.0153556881241978526 7e-01, 8.6513702568784012214 6e-01,
        -4.8795713146419933403 7e-01, 8.7286759468617580726 1e-01,
        -4.7425956637550770178 5e-01, 8.8038506558284790326 5e-01,
        -4.6044621761648452107 4e-01, 8.8768760309169181255 1e-01,
        -4.4652045752219976515 5e-01, 8.9477342440092921815 9e-01,
        -4.3248568585718766277 3e-01, 9.0164079961403587049 1e-01,
        -4.1834532901560084194 9e-01, 9.0828805215681918117 1e-01,
        -4.0410283915886024974 6e-01, 9.1471355919968100334 2e-01,
        -3.8976169338768829053 5e-01, 9.2091575204660369724 5e-01,
        -3.7532539288714489300 6e-01, 9.2689311652105299543 8e-01,
        -3.6079746209183716221 2e-01, 9.3264419332781423044 3e-01,
        -3.4618144775443082661 3e-01, 9.3816578437160809557 e-01,
        -3.3148091818918684614 6e-01, 9.4346192338453893633 2e-01,
        -3.1669946230523471353 3e-01, 9.4852593563675169363 6e-01,
        -3.0184068880858827554 9e-01, 9.5335878879399670502 e-01,
        -2.8690822522343317757 5e-01, 9.5795807335140703564 5e-01,
        -2.7190571709267269406 9e-01, 9.6232389610375956845 4e-01,
        -2.5683682710636551727 0e-01, 9.6645478127118544797 7e-01,
        -2.4170523408433014500 6e-01, 9.7034972036696087727 1e-01,
        -2.2651463218853262748 8e-01, 9.7400776249704179576 8e-01,
        -2.1126872999172152667 3e-01, 9.7742801460142580971 5e-01,
        -1.9597124957308914572 4e-01, 9.8060964167234354604 8e-01,
        -1.8062592560285750664 7e-01, 9.8355186659801457391 e-01,
        -1.6523650438810191798 4e-01, 9.8625397216822441315 3e-01,
        -1.4980674303727931073 7e-01, 9.8871529761633736299 6e-01,
        -1.3434040853823514538 6e-01, 9.9093524240173236350 4e-01,
        -1.1884127673230783038 e-01, 9.9291326456273709677 4e-01,
        -1.0331313154973309487 2e-01, 9.9464888118842598147 8e-01,
        -8.7759763960557610196 2e-02, 9.9614166855402008771 1e-01,
        -7.2184971062410077657 9e-02, 9.9739126221995699772 5e-01,
        -5.6592555240632259894 2e-02, 9.9839735711355726000 e-01,
        -4.0986323147317968440 5e-02, 9.9915970761178296566 4e-01,
        -2.5370084850008287058 5e-02, 9.9967812759642959885 5e-01,
        -9.7476528133329000402 9e-03, 9.9995249050373924145 4e-01,
        5.8771589965862479354 5e-03, 9.9998272935192678012 6e-01,
        2.1500535957733660913 4e-02, 9.9976883675854300026 5e-01,
        3.7118663884409153208 6e-02, 9.9931086494215415339 0e-01,
        5.2727729811954456018 4e-02, 9.9860892571059944877 7e-01,
        6.8323923035028866219 e-02, 9.9766319043138096400 7e-01,
        8.3903435925953492952 e-02, 9.9647388999402208842 3e-01,
        9.9462465019891019291 1e-02, 9.9504131474636114962 4e-01,
        1.1499721177297986263 2e-01, 9.9336581443315252748 5e-01,
        1.3050388360153000744 1e-01, 9.9144779810382266394 0e-01,
        1.4597869479814026827 4e-01, 9.8928773401120839725 6e-01,
        1.6141786739019647889 4e-01, 9.8688614950621378341 1e-01,
        1.7681763208696590905 5e-01, 9.8424363090809907639 3e-01,
        1.9217422931651081952 1e-01, 9.8136082334002161520 2e-01,
        2.0748390997241900057 8e-01, 9.7823843059990898876 e-01,
        2.2274293638447961729 6e-01, 9.7487721498187640545 3e-01,
        2.3794758333776275249 8e-01, 9.7127799706557671477 5e-01,
        2.5309413876141773069 9e-01, 9.6744165556617223167 3e-01,
        2.6817890491331314306 6e-01, 9.6336912705333055395 6e-01,
        2.8319819900889836583 6e-01, 9.5906140579116017086 4e-01,
```



```
        2.9814835435589576162 5e-01, 9.5451954343264822089 3e-01,
        3.1302572098467484895 7e-01, 9.4974464884095266548 1e-01,
        3.2782666695308820825 6e-01, 9.4473788768865885057 1e-01,
        3.4254757872312369126 9e-01, 9.3950048233671779041 0e-01,
        3.5718486242209529502 0e-01, 9.3403371141330271409 9e-01,
        3.7173494455292199756 3e-01, 9.2833890955740727690 7e-01,
        3.8619427292218388991 8e-01, 9.2241746707339933088 e-01,
        4.0055931749265044628 0e-01, 9.1627082959669847728 2e-01,
        4.1482657125514488250 0e-01, 9.0990049773626358042 8e-01,
        4.2899255106913575241 7e-01, 9.0330802671469434539 4e-01,
        4.4305379849304421524 5e-01, 8.9649502599896502275 1e-01,
        4.5700688068885580994 7e-01, 8.8946315887901838959 1e-01,
        4.7084839122735999694 7e-01, 8.8221414207583803701 6e-01,
        4.8457495085152435532 2e-01, 8.7474974535991878443 8e-01,
        4.9818320847084640590 2e-01, 8.6707179102868592313 1e-01,
        5.1166984180138519455 7e-01, 8.5918215355705884750 4e-01,
        5.2503155827309566650 e-01, 8.5108275908828334710 4e-01,
        5.3826509583892634403 0e-01, 8.4277558495812598948 8e-01,
        5.5136722367484081175 3e-01, 8.3426265927290432777 9e-01,
        5.6433474312905307357 4e-01, 8.2554606031248534137 0e-01,
        5.7716448833973155174 7e-01, 8.1662791612798555378 9e-01,
        5.8985327114563730227 e-01, 8.0751040395271656002 8e-01,
        6.0239816166790927089 e-01, 7.9819574968745821141 9e-01,
        6.1479592931080073725 5e-01, 7.8868622740787663882 9e-01,
        6.2704360344099663304 7e-01, 7.7898415862181058511 0e-01,
        6.3913819378390790415 5e-01, 7.6909191209285499013 5e-01,
        6.5107674773277257607 2e-01, 7.5901190277999963651 5e-01,
        6.6285635063440673242 5e-01, 7.4874659159400280827 9e-01,
        6.7447412665261075037 6e-01, 7.3829848467689407343 1e-01,
        6.8592723948851241910 8e-01, 7.2767013277148384631 2e-01,
        6.9721289302888467265 3e-01, 7.1686413063724496730 3e-01,
        7.0832833205651568597 7e-01, 7.0588311639111644968 4e-01,
        7.1927084285818132514 1e-01, 6.9472977092829513168 2e-01,
        7.3003775398209846958 5e-01, 6.8340681717464091260 4e-01,
        7.4062643690159323611 e-01, 6.7191701940228476530 6e-01,
        7.5103430648362201616 0e-01, 6.6026318274205242353 5e-01,
        7.6125882180779735897 1e-01, 6.4844815229885899299 2e-01,
        7.7129748671370212953 5e-01, 6.3647481253316418037 3e-01,
        7.8114785042413459326 1e-01, 6.2434608653995249394 3e-01,
        7.9080750803152544126 1e-01, 6.1206493547741225302 9e-01,
        8.0027410132632414985 2e-01, 5.9963435754328175963 9e-01,
        8.0954531923643069379 9e-01, 5.8705738740125357200 1e-01,
        8.1861889824964528816 8e-01, 5.7433709564030155370 1e-01,
        8.2749262316867178146 4e-01, 5.6147658775894704330 5e-01,
        8.3616432765427695095 2e-01, 5.4847900338889088445 2e-01,
        8.4463189460347687376 2e-01, 5.3534751574892092129 7e-01,
        8.5289325679355443288 2e-01, 5.2208533068463436333 0e-01,
        8.6094639732888444960 7e-01, 5.0869568597189285252 8e-01,
        8.6878935015379210593 5e-01, 4.9518185049469615188 8e-01,
        8.7642020050937829989 1e-01, 4.8154712348751615991 2e-01,
        8.8383708545414108037 6e-01, 4.6779483363535484530 3e-01,
        8.9103819424076324828 8e-01, 4.5392833840173479886 8e-01,
        8.9802176886999907079 5e-01, 4.3995102299642169230 2e-01,
        9.0478610429355543765 0e-01, 4.2586630000188019362 6e-01,
        9.1132954920060382786 3e-01, 4.1167760778772533036 8e-01,
        9.1765050606466636029 5e-01, 3.9738841039877059735 4e-01,
        9.2374743172307749450 3e-01, 3.8300219631879173221 0e-01,
        9.2961883769702957636 1e-01, 3.6852247773890778326 2e-01,
        9.3526329056056201061 2e-01, 3.5395278969070120833 6e-01,
        9.4067941230483764769 6e-01, 3.3929668914657162837 0e-01,
        9.4586588066181128528 5e-01, 3.2455775418229504403 2e-01,
        9.5082142943115011135 5e-01, 3.0973958309280463785 4e-01,
        9.5554484878494583277 6e-01, 2.9484579352698081500 3e-01,
        9.6003498563746725302 8e-01, 2.7988002135212874943 4e-01,
        9.6429074358031818814 4e-01, 2.6484592095276271450 6e-01,
        9.6831108383186226262 8e-01, 2.4974716200262259135 9e-01,
```

```
        9.72095024823529496594e-01, 2.34587430851146777622e-01,
        9.75641642748477533331e-01, 2.19370428579268944569e-01,
        9.78950071770562035844e-01, 2.04099870113656461923e-01,
        9.82019504170923540620e-01, 1.88779483598969205493e-01,
        9.84849190595408208182e-01, 1.73413009268535894813e-01,
        9.87438440210647860873e-01, 1.58004198660550820854e-01,
        9.89786620894844482166e-01, 1.42556813578185059832e-01,
        9.91893159367450705233e-01, 1.27074625319365142051e-01,
        9.93757541353338824663e-01, 1.11561413595234956708e-01,
        9.95379311685924861308e-01, 9.60209657713066710993e-02,
        9.96758074438687136087e-01, 8.04570757688883031467e-02,
        9.97893492999770703733e-01, 6.48735433648924275651e-02,
        9.98785290176304019205e-01, 4.92741730264058125366e-02,
        9.99433248251151762354e-01, 3.36627730609296640929e-02,
        9.99837209032161888800e-01, 1.80431547900308138221e-02,
        9.99997073896832011641e-01, 2.41913161566219324719e-03,
        9.99912803818512774257e-01, -1.32054821873487954892e-02,
        9.99584419370001642235e-01, -2.88268720595330242562e-02,
        9.99012000721555049054e-01, -4.44412242666163900817e-02,
        9.98195687620527460915e-01, -6.00447267941353959864e-02,
        9.97135679355775073063e-01, -7.56335702958476074897e-02,
        9.95832234722008102779e-01, -9.12039488650100010902e-02,
        9.94285671925894676271e-01, -1.06752061351864019345e-01,
        9.92496368545773943737e-01, -1.22274111828511194977e-01,
        9.90464761404806215417e-01, -1.37766310886661608182e-01
};

const double _TBL_sincosx[] = {
        1.64062500000167837966e-01, 1.79687499999472477530e-01,
        1.95312499999996669331e-01, 2.10937500000106192832e-01,
        2.26562499999874683576e-01, 2.42187499999999750200e-01,
        2.57812499999549193941e-01, 2.73437500000180466753e-01,
        2.89062500000347444296e-01, 3.04687500000159650071e-01,
        3.20312500001052657961e-01, 3.35937499999853450561e-01,
        3.51562499998759436792e-01, 3.67187499998127386824e-01,
        3.82812499999808708573e-01, 3.98437499999694078046e-01,
        4.14062499999775512904e-01, 4.29687499999869215728e-01,
        4.45312499999981514787e-01, 4.60937499992721433362e-01,
        4.76562499999932387418e-01, 4.92187500000263733479e-01,
        5.07812500002462252624e-01, 5.23437499998664290679e-01,
        5.39062500000133337785e-01, 5.54687499999937494444e-01,
        5.70312499999814259688e-01, 5.85937500002074562744e-01,
        6.01562499999652833260e-01, 6.17187499994191313140e-01,
        6.32812500000347721851e-01, 6.48437500005533351555e-01,
        6.64062499997531863194e-01, 6.79687499999813815599e-01,
        6.95312500005013212068e-01, 7.10937499998876987289e-01,
        7.26562500001548428052e-01, 7.42187500000339617223e-01,
        7.57812499998633315457e-01, 7.73437500000337285755e-01,
        7.89062499996497468402e-01, 8.04687500001799671152e-01,
        8.20312500001350475287e-01, 8.35937499996779354028e-01,
        8.51562500000668243239e-01, 8.67187499999485522650e-01,
        8.82812500000538014078e-01, 8.98437500000525690602e-01,
        9.14062500000757727214e-01, 9.29687500002357114504e-01,
        9.45312499999430455588e-01, 9.60937500000796696042e-01,
        9.76562500001389000026e-01, 9.92187499998313238159e-01,
        1.00781250000027000624e+00, 1.02343750000073119288e+00,
        1.03906249999567279474e+00, 1.05468750000121480603e+00,
        1.07031249999813948826e+00, 1.08593749999936250994e+00,
        1.10156249999885291757e+00, 1.11718750000074029671e+00,
        1.13281249999926680871e+00, 1.14843749999650057703e+00,
        1.16406249999956079577e+00, 1.17968749999995736744e+00,
        1.19531250000235189646e+00, 1.21093750000001554312e+00,
        1.22656249999714606069e+00, 1.24218750000679789558e+00,
        1.25781249999789324079e+00, 1.27343750000030864200e+00,
        1.28906250000041366910e+00, 1.30468750000013344881e+00,
        1.32031249999823008245e+00, 1.33593749999817146268e+00,
```



```
        1.35156249999504352033e+00, 1.36718750000051336713e+00,
        1.38281250000255573340e+00, 1.39843749999889488400e+00,
        1.41406250000066702199e+00, 1.42968750000377853304e+00,
        1.44531250000268074452e+00, 1.46093749999857935862e+00,
        1.47656250000000177636e+00, 1.49218750000007549517e+00,
        1.50781249999986965982e+00, 1.52343749999979238829e+00,
        1.53906250000026356695e+00, 1.55468750000024247271e+00,
        1.57031250000686006807e+00, 1.58593749999970379250e+00,
        1.60156249999876076906e+00, 1.61718749999920530236e+00,
        1.63281249999894950697e+00, 1.64843749999433342168e+00,
        1.66406250000158717484e+00, 1.67968749999775224246e+00,
        1.69531250000185917948e+00, 1.71093749999863442568e+00,
        1.72656249999789279670e+00, 1.74218750000263478128e+00,
        1.75781250000296740410e+00, 1.77343749999920641258e+00,
        1.78906249999844191301e+00, 1.80468749999888578017e+00,
        1.82031250003296385387e+00, 1.83593749999912847493e+00,
        1.85156249999896371783e+00, 1.86718749999873900869e+00,
        1.88281249999986122212e+00, 1.89843750000025601743e+00,
        1.91406250000089750429e+00, 1.92968749999936717288e+00,
        1.94531249999502553472e+00, 1.96093749999814637164e+00,
        1.97656250000163713487e+00, 1.99218750000058819616e+00,
        2.00781250000015099033e+00, 2.02343750000025890401e+00,
        2.03906249999571986820e+00, 2.05468749999347455315e+00,
        2.07031249999880184731e+00, 2.08593749999950617280e+00,
        2.10156249999859534583e+00, 2.11718749999749178414e+00,
        2.13281250000269562150e+00, 2.14843750000770983277e+00,
        2.16406250000204325445e+00, 2.17968750000288169488e+00,
        2.19531250000207567297e+00, 2.21093749999685940111e+00,
        2.22656249999882449586e+00, 2.24218750000040500936e+00,
        2.25781249999956967756e+00, 2.27343749999970867748e+00,
        2.28906249999833111275e+00, 2.30468749999696020936e+00,
        2.32031250000405675493e+00, 2.33593750000527755617e+00,
        2.35156250000277511347e+00, 2.36718749998901101250e+00,
        2.38281250000068833828e+00, 2.39843750000151390012e+00,
        2.41406250000618571860e+00, 2.42968749999278221807e+00,
        2.44531250000394617672e+00, 2.46093750000379341003e+00,
        2.47656250000329514194e+00, 2.49218749999781508109e+00,
        2.50781249999807354101e+00, 2.52343750000954214485e+00,
        2.53906250000098099306e+00, 2.55468750001107025582e+00,
        2.57031250000341415785e+00, 2.58593750002171240965e+00,
        2.60156250000635891340e+00, 2.61718750000451771953e+00,
        2.63281250000028421709e+00, 2.64843750001994493459e+00,
        2.66406250000455235849e+00, 2.67968749999316235844e+00,
        2.69531249997396704643e+00, 2.71093749999957500663e+00,
        2.72656249999638511383e+00, 2.74218749999314947985e+00,
        2.75781249999954258811e+00, 2.77343750000063726802e+00,
        2.78906249999834177089e+00, 2.80468750000019895197e+00,
        2.82031249999983835153e+00, 2.83593749999777511306e+00,
        2.85156249999855315735e+00, 2.86718750000235678144e+00,
        2.88281249999902611236e+00, 2.89843749999328359479e+00,
        2.91406250000365130148e+00, 2.92968749999994892974e+00,
        2.94531249999847322130e+00, 2.96093749999701350006e+00,
        2.97656250000468292072e+00, 2.99218750000308997272e+00,
        3.00781249999819877416e+00, 3.02343749999709299203e+00,
        3.03906249999948618878e+00, 3.05468750000752597984e+00,
        3.07031250000433075797e+00, 3.08593749999511279825e+00,
        3.10156249999957589480e+00, 3.11718749999961186603e+00,
        3.13281249999836441944e+00, 3.14843750000262057043e+00,
        3.16406249999657873673e+00, 3.17968750000540190115e+00,
        3.19531250000325739435e+00, 3.21093750000270583556e+00,
        3.22656250000035882408e+00, 3.24218749999618305324e+00,
        3.25781250000001199041e+00, 3.27343750000431255032e+00,
        3.28906249999634914261e+00, 3.30468749999773381276e+00,
        3.32031250000108801856e+00, 3.33593750000042854609e+00,
        3.35156249999819699781e+00, 3.36718749999951061369e+00,
        3.38281250000727817806e+00, 3.39843750000385558252e+00,
```

```
        3.41406250000184297022e+00, 3.42968750000183808524e+00,
        3.44531249999830135877e+00, 3.46093749998354383024e+00,
        3.47656249999984101606e+00, 3.49218750000081934459e+00,
        3.50781249999577759979e+00, 3.52343749999866640010e+00,
        3.53906249999683852892e+00, 3.55468750000498978636e+00,
        3.57031249999826005848e+00, 3.58593750001092637092e+00,
        3.60156250000782085507e+00, 3.61718749999987299049e+00,
        3.63281250000544186918e+00, 3.64843749999226352188e+00,
        3.66406250000062438943e+00, 3.67968749999757616109e+00,
        3.69531250001872235700e+00, 3.71093750000574784664e+00,
        3.72656249999563016218e+00, 3.74218749999581179466e+00,
        3.75781250000033528735e+00, 3.77343749999415045693e+00,
        3.78906249995283994636e+00, 3.80468750000592104143e+00,
        3.82031249998920063859e+00, 3.83593750000164934733e+00,
        3.85156250000057731597e+00, 3.86718750000405053768e+00,
        3.88281249997192157153e+00, 3.89843749998371702503e+00,
        3.91406249999986277643e+00, 3.92968749999597033451e+00,
        3.94531249999519229021e+00, 3.96093749997563104870e+00,
        3.97656250000223510099e+00, 3.99218750000022870594e+00,
        4.00781250004454392410e+00, 4.02343749999355093649e+00,
        4.03906249999698196973e+00, 4.05468749999250022142e+00,
        4.07031249994990851349e+00, 4.08593749999590372113e+00,
        4.10156249999066258027e+00, 4.11718749999303490483e+00,
        4.13281249999853184107e+00, 4.14843749998088373587e+00,
        4.16406249999834177089e+00, 4.17968749999662758654e+00,
        4.19531249999891109326e+00, 4.21093749999872102308e+00,
        4.22656249999120881000e+00, 4.24218750000338129524e+00,
        4.25781250000494537744e+00, 4.27343749997698019172e+00,
        4.28906250000330668826e+00, 4.30468749999959232611e+00,
        4.32031250000562039304e+00, 4.33593749999550670537e+00,
        4.35156250000948219281e+00, 4.36718750000763922259e+00,
        4.38281249999987476684e+00, 4.39843750000314237525e+00,
        4.41406250000408473255e+00, 4.42968750000079314333e+00,
        4.44531249998868371875e+00, 4.46093750000322319949e+00,
        4.47656249999480770896e+00, 4.49218749997964028609e+00,
        4.50781250000320810045e+00, 4.52343749999724753508e+00,
        4.53906249999181721222e+00, 4.55468750000258193467e+00,
        4.57031249999976196818e+00, 4.58593750000821920310e+00,
        4.60156250004601385939e+00, 4.61718750000444977388e+00,
        4.63281249999695177166e+00, 4.64843749999638600201e+00,
        4.66406250000544897460e+00, 4.67968749999663469197e+00,
        4.69531249998381028377e+00, 4.71093749999796340688e+00,
        4.72656250000119992905e+00, 4.74218750001258992910e+00,
        4.75781250000492050845e+00, 4.77343750000340971695e+00,
        4.78906250000747402140e+00, 4.80468749998990762862e+00,
        4.82031250001256594828e+00, 4.83593750000031530334e+00,
        4.85156250000026023628e+00, 4.86718750000094679820e+00,
        4.88281250000185362836e+00, 4.89843749997600141910e+00,
        4.91406249999471889112e+00, 4.92968749998860822359e+00,
        4.94531250000475353090e+00, 4.96093749999659205940e+00,
        4.97656250000856825721e+00, 4.99218750002637179364e+00,
        5.00781249999760014191e+00, 5.02343749998691091463e+00,
        5.03906249999699618058e+00, 5.05468750000537525580e+00,
        5.07031250000353406193e+00, 5.08593749999286881547e+00,
        5.10156249998831601289e+00, 5.11718750000479172257e+00,
        5.13281250001085087575e+00, 5.14843750000346744855e+00,
        5.16406250000581845683e+00, 5.17968750000350119933e+00,
        5.19531249999482636071e+00, 5.21093750000432454073e+00,
        5.22656250000434585701e+00, 5.24218750001077093970e+00,
        5.25781249998869881779e+00, 5.27343750002139977084e+00,
        5.28906249999702104958e+00, 5.30468749998945909851e+00,
        5.32031249999385913441e+00, 5.33593749999546851370e+00,
        5.35156250001908162517e+00, 5.36718749999724487054e+00,
        5.38281249999679634044e+00, 5.39843750001770139590e+00,
        5.41406249999678212959e+00, 5.42968749999906563630e+00,
        5.44531250000517097476e+00, 5.46093749999811794993e+00,
```



```
        5.47656250001082511858e+00, 5.49218749999457500621e+00,
        5.50781250001214406353e+00, 5.52343750001415045858e+00,
        5.53906250000498356911e+00, 5.55468750000498889818e+00,
        5.57031250000316013882e+00, 5.58593750000908428888e+00,
        5.60156250002763478335e+00, 5.61718749999503863535e+00,
        5.63281250000129496414e+00, 5.64843750001081890133e+00,
        5.66406250000738609174e+00, 5.67968750000023270275e+00,
        5.69531249998335997731e+00, 5.71093749999160404940e+00,
        5.72656250000217958984e+00, 5.74218750000474997819e+00,
        5.75781250001663868918e+00, 5.77343749999750688318e+00,
        5.78906249999925304195e+00, 5.80468749999988631316e+00,
        5.82031249999487254598e+00, 5.83593749999551025809e+00,
        5.85156249999513455862e+00, 5.86718749999803179662e+00,
        5.88281250000295141689e+00, 5.89843750000985433957e+00,
        5.91406249999845634591e+00, 5.92968750000455990801e+00,
        5.94531250000243982612e+00, 5.96093750000733901828e+00,
        5.97656249999234212567e+00, 5.99218749999141753193e+00,
        6.00781250000843591863e+00, 6.02343749999880984092e+00,
        6.03906249999745359247e+00, 6.05468750000370548037e+00,
        6.07031250001220445967e+00, 6.08593750001188915633e+00,
        6.10156249999290700714e+00, 6.11718749998957456171e+00,
        6.13281249999975663911e+00, 6.14843749999015098950e+00,
        6.16406250000358646446e+00, 6.17968750000026467717e+00,
        6.19531249998414246249e+00, 6.21093749998937294521e+00,
        6.22656249999281197205e+00, 6.24218750000707967018e+00,
        6.25781250000234834374e+00, 6.27343749999462829692e+00,
        6.28906250001052136156e+00, 6.30468750000171862524e+00,
        6.32031250000594013727e+00, 6.33593750000045385917e+00,
        6.35156250000499689179e+00, 6.36718749999230215764e+00,
        6.38281249999868105505e+00, 6.39843749999853628196e+00,
        6.41406249999377564563e+00, 6.42968750000876010375e+00,
        6.44531250002396838283e+00, 6.46093750000062527761e+00,
        6.47656249999929212180e+00, 6.49218750000642064180e+00,
        6.50781249999003996720e+00, 6.52343750000912248055e+00,
        6.53906249998720845440e+00, 6.55468749999868371958e+00,
        6.57031249998638067211e+00, 6.58593750000546407364e+00,
        6.60156249994729282804e+00, 6.61718749997319211076e+00,
        6.63281249997879296387e+00, 6.64843749999244426618e+00,
        6.66406249999900524017e+00, 6.67968749999884092716e+00,
        6.69531249999227373593e+00, 6.71093749999063504674e+00,
        6.72656249999940136775e+00, 6.74218749999563193853e+00,
        6.75781249999463895506e+00, 6.77343750001427569174e+00,
        6.78906249997048583 11e+00, 6.80468750000215738538e+00,
        6.82031250000341859874e+00, 6.83593749999844302323e+00,
        6.85156250001598987609e+00, 6.86718750000203925765e+00,
        6.88281250000989430760e+00, 6.89843750000604671868e+00,
        6.91406249999750777135e+00, 6.92968749999960120789e+00,
        6.94531249995244071016e+00, 6.96093750002739852789e+00,
        6.97656249994302335544e+00, 6.99218749999911892701e+00,
        7.00781250000804245559e+00, 7.02343750000080380147e+00,
        7.03906249999665778461e+00, 7.05468749999575539533e+00,
        7.07031250001700328767e+00, 7.08593750000647215614e+00,
        7.10156249997034372257e+00, 7.11718749999698641062e+00,
        7.13281250000188915550e+00, 7.14843750000192734717e+00,
        7.16406250000996358551e+00, 7.17968750005667022407e+00,
        7.19531250000950883816e+00, 7.21093749995827248966e+00,
        7.22656250000638511466e+00, 7.24218750002578115499e+00,
        7.25781249999116351290e+00, 7.27343749999614619384e+00,
        7.28906249998626343256e+00, 7.30468749998397015588e+00,
        7.32031249998488320330e+00, 7.33593749999550048813e+00,
        7.35156249998663557932e+00, 7.36718750000183675297e+00,
        7.38281249999652722238e+00, 7.39843750007829115134e+00,
        7.41406250000048494542e+00, 7.42968750000089794838e+00,
        7.44531249991165467557e+00, 7.46093749999333422096e+00,
        7.47656250000219557705e+00, 7.49218750000104360964e+00,
        7.50781249999853983468e+00, 7.52343749999573585541e+00,
```

```
        7.53906249999752819946e+00, 7.55468749999868549594e+00,
        7.57031250024692781153e+00, 7.58593750000690736357e+00,
        7.60156249999365662973e+00, 7.61718749999451283372e+00,
        7.63281249995806998498e+00, 7.64843749997276400876e+00,
        7.66406249998022381931e+00, 7.67968750000013145041e+00,
        7.69531249999808597551e+00, 7.71093750001158539931e+00,
        7.72656249999979038989e+00, 7.74218750000620037355e+00,
        7.75781249999318234245e+00, 7.77343750001715427800e+00,
        7.78906250000142730272e+00, 7.80468749997501465288e+00,
        7.82031249999300381859e+00, 7.83593750003314948316e+00,
        7.85156249999927524641e+00, 7.86718749999776001403e+00,
        7.88281249999449951105e+00, 7.89843749999351540936e+00,
        7.91406250000050803806e+00, 7.92968750004068656523e+00,
        7.94531249999952127183e+00, 7.96093750001230571200e+00,
        7.97656249999947331020e+00, 7.99218750003934363946e+00
};
```

```
**********************************************************
    4960 Thu Jun  6 20:37:32 2019
new/usr/src/lib/libm/common/C/_TBL_tan.c
11210 libm should be cstyle(10NBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#include "libm_protos.h"

const double _TBL_tan_hi[] = {
        1.57534107325271622e-01, 1.61539784049521462e-01,
        1.65550519273933966e-01, 1.69566445219766521e-01,
        1.73587694767981526e-01, 1.77614401477446726e-01,
        1.81646699603321415e-01, 1.85684724115634414e-01,
        1.89728610718059132e-01, 1.93778495866891859e-01,
        1.97834516790238668e-01, 2.01896811507417145e-01,
        2.05965518848578860e-01, 2.10040778474558987e-01,
        2.14122730896958657e-01, 2.18211517498467428e-01,
        2.22307280553431325e-01, 2.26410163248673829e-01,
        2.30520309704576154e-01, 2.34637864996423667e-01,
        2.38762975176025932e-01, 2.42895787293616550e-01,
        2.47036449420041271e-01, 2.51185110669240763e-01,
        2.55341921221036272e-01, 2.63680596419996804e-01,
        2.72053698658770882e-01, 2.80462470145251386e-01,
        2.88908172440514699e-01, 2.97392087269024608e-01,
        3.05915517353059274e-01, 3.14479787272571532e-01,
        3.23086244351745544e-01, 3.31736259573572778e-01,
        3.40431228523830398e-01, 3.49172572365910372e-01,
        3.57961738848017019e-01, 3.66800203344323394e-01,
        3.75689469931754838e-01, 3.84631072504149241e-01,
        3.93626575925632771e-01, 4.02677577225140193e-01,
        4.11785706834108478e-01, 4.20952629869475847e-01,
        4.30180047464230053e-01, 4.39469698147866239e-01,
        4.48823359279239720e-01, 4.58242848534432368e-01,
        4.67730025452391784e-01, 4.77286793041252266e-01,
        4.86915099448406330e-01, 4.96616939697565651e-01,
        5.06394357496229852e-01, 5.16249447117175131e-01,
```



```
        5.26184355357779188e-01, 5.36201283581215993e-01,
        5.46302489843790484e-01, 5.66767065580586427e-01,
        5.87597367591443209e-01, 6.08813740324380737e-01,
        6.30437673835884782e-01, 6.52491897928808018e-01,
        6.75000485144242934e-01, 6.97988963623599301e-01,
        7.21484440990904474e-01, 7.45515740559391960e-01,
        7.70113551344208669e-01, 7.95310593568674173e-01,
        8.21141801589894138e-01, 8.47644526446552637e-01,
        8.74858760554482306e-01, 9.02827387452673547e-01,
        9.31596459944072475e-01, 9.61215510494370373e-01,
        9.91737898363268644e-01,
 1.57534107325271622e-01, 1.61539784049521462e-01, 1.65550519273933966e-01,
 1.69566445219766521e-01, 1.73587694767981526e-01, 1.77614401477446726e-01,
 1.81646699603321415e-01, 1.85684724115634414e-01, 1.89728610718059132e-01,
 1.93778495866891859e-01, 1.97834516790238668e-01, 2.01896811507417145e-01,
 2.05965518848578860e-01, 2.10040778474558987e-01, 2.14122730896958657e-01,
 2.18211517498467428e-01, 2.22307280553431325e-01, 2.26410163248673829e-01,
 2.30520309704576154e-01, 2.34637864996423667e-01, 2.38762975176025932e-01,
 2.42895787293616550e-01, 2.47036449420041271e-01, 2.51185110669240763e-01,
 2.55341921221036272e-01, 2.63680596419996804e-01, 2.72053698658770882e-01,
 2.80462470145251386e-01, 2.88908172440514699e-01, 2.97392087269024608e-01,
 3.05915517353059274e-01, 3.14479787272571532e-01, 3.23086244351745544e-01,
 3.31736259573572778e-01, 3.40431228523830398e-01, 3.49172572365910372e-01,
 3.57961738848017019e-01, 3.66800203344323394e-01, 3.75689469931754838e-01,
 3.84631072504149241e-01, 3.93626575925632771e-01, 4.02677577225140193e-01,
 4.11785706834108478e-01, 4.20952629869475847e-01, 4.30180047464230053e-01,
 4.39469698147866239e-01, 4.48823359279239720e-01, 4.58242848534432368e-01,
 4.67730025452391784e-01, 4.77286793041252266e-01, 4.86915099448406330e-01,
 4.96616939697565651e-01, 5.06394357496229852e-01, 5.16249447117175131e-01,
 5.26184355357779188e-01, 5.36201283581215993e-01, 5.46302489843790484e-01,
 5.66767065580586427e-01, 5.87597367591443209e-01, 6.08813740324380737e-01,
 6.30437673835884782e-01, 6.52491897928808018e-01, 6.75000485144242934e-01,
 6.97988963623599301e-01, 7.21484440990904474e-01, 7.45515740559391960e-01,
 7.70113551344208669e-01, 7.95310593568674173e-01, 8.21141801589894138e-01,
 8.47644526446552637e-01, 8.74858760554482306e-01, 9.02827387452673547e-01,
 9.31596459944072475e-01, 9.61215510494370373e-01, 9.91737898363268644e-01,
};

#endif /* ! codereview */
const double _TBL_tan_lo[] = {
        -1.10615392752930551e-17, 1.42255435911932711e-17,
        1.02781342487141920e-17, -1.04735896510580927e-17,
        -5.46679990560150911e-18, 1.50201543247778489e-18,
        1.22522327805930836e-17, -2.52772423968968903e-18,
        9.78955701743985001e-19, 4.61515122717816178e-18,
        7.14813042382104539e-19, -1.25529090642919992e-17,
        1.19416304006222131e-17, -5.91325462642753544e-18,
        7.53213214053688138e-18, 4.77223821731568090e-18,
        6.32882137760769522e-18, 8.33823681661647871e-18,
        -1.25419320906151988e-17, 1.16585041935775587e-17,
        -1.19653634178542542e-17, -7.22806346068389604e-18,
        -6.16674472236513534e-18, 4.26199277415660669e-18,
        -5.58935834356478328e-18, -4.56998635843850688e-18,
        1.78004627511465564e-18, 1.74249040881549088e-17,
        2.70817328270223006e-17, -1.80870634839170844e-17,
        -1.00676145758650168e-17, -1.53577462986005684e-17,
        -2.38939880909534397e-17, -1.08193046058071237e-17,
        -1.06856311222117164e-17, -1.96951245902998606e-17,
        -2.08660034657941102e-17, 2.82596474303348100e-17,
        2.34797942068937341e-18, -1.76131026613802985e-17,
        -1.29729310968305823e-17, 1.87495311063417555e-17,
        -2.29163073231136327e-18, -2.51936954463539765e-17,
        -4.11327516430776285e-18, 1.50393242431203736e-18,
        -1.09029595007501330e-17, -6.87284752683418342e-19,
        1.55195027932634982e-17, -4.62284921534513474e-18,
        -5.45294879014110259e-18, -2.56576334605328725e-17,
```

```
        -4.00960685506800741e-17, 1.35860113023765056e-17,
        -4.34857062258506890e-17, 3.85791583096984630e-17,
        2.90965762168371759e-17, 1.90815918857458480e-17,
        1.21159907937263400e-17, -1.52112721227855650e-17,
        -1.51838757657007437e-17, -2.51352280752587451e-17,
        -2.66690480643161193e-17, -4.59728584599455591e-17,
        -5.42439848134543255e-17, 3.56284233494755594e-17,
        3.61475127591663133e-17, 1.22197541073075113e-17,
        -1.61356193051149559e-17, 1.66243632690603545e-17,
        4.30578558405427098e-17, -4.43234026650131250e-17,
        -1.35473813965930355e-17, 4.30118334112910435e-17,
        3.62593428168003066e-17,
 -1.10615392752930551e-17, 1.42255435911932711e-17, 1.02781342487141920e-17,
 -1.04735896510580927e-17,-5.46679990560150911e-18, 1.50201543247778489e-18,
  1.22522327805930836e-17,-2.52772423968968903e-18, 9.78955701743985001e-19,
  4.61515122717816178e-18, 7.14813042382104539e-19,-1.25529909642919992e-17,
  1.19416304006222131e-17,-5.91325462642753544e-18, 7.53213214053688138e-18,
  4.77223821731568090e-18, 6.32882137760769522e-18, 8.33823681661647871e-18,
 -1.25419320906151988e-17, 1.16585041935775587e-17,-1.19653634178542542e-17,
 -7.22806346068389604e-18,-6.16674472236513534e-18, 4.26199277415660669e-18,
 -5.58935834356478328e-18,-4.56998635843850688e-18, 1.78004627511465564e-18,
  1.74249040881549088e-17, 2.70817328270223006e-17,-1.80870634839170844e-17,
 -1.00676145758650168e-17,-1.53577462986005684e-17,-2.38939880909534397e-17,
 -1.08193046058071237e-17,-1.06856311222117164e-17,-1.96951245902998606e-17,
 -2.08660034657941102e-17, 2.82596474303348100e-17, 2.34797942068937341e-18,
 -1.76131026613802985e-17,-1.29729310968305823e-17, 1.87495311063417555e-17,
 -2.29163073231136327e-18,-2.51936954463539765e-17,-4.11327516430776285e-18,
  1.50393242431203736e-18,-1.09029595007501330e-17,-6.87284752683418342e-19,
  1.55195027932634982e-17,-4.62284921534513474e-18,-5.45294879014110259e-18,
 -2.56576334605328725e-17,-4.00960685506800741e-17, 1.35860113023765056e-17,
 -4.34857062258506890e-17, 3.85791583096984630e-17, 2.90965762168371759e-17,
  1.90815918857458480e-17, 1.21159907937263400e-17,-1.52112721227855650e-17,
 -1.51838757657007437e-17,-2.51352280752587451e-17,-2.66690480643161193e-17,
 -4.59728584599455591e-17,-5.42439848134543255e-17, 3.56284233494755594e-17,
  3.61475127591663133e-17, 1.22197541073075113e-17,-1.61356193051149559e-17,
  1.66243632690603545e-17, 4.30578558405427098e-17,-4.43234026650131250e-17,
 -1.35473813965930355e-17, 4.30118334112910435e-17, 3.62593428168003066e-17,
};
```

*_____unchanged_portion_omitted_*

```
**********************************************************
    3636 Thu Jun  6 20:37:32 2019
new/usr/src/lib/libm/common/C/__cos.c
11210 libm should be cstyle(10NBLD) clean
**********************************************************
   1 /*
   2  * CDDL HEADER START
   3  *
   4  * The contents of this file are subject to the terms of the
   5  * Common Development and Distribution License (the "License").
   6  * You may not use this file except in compliance with the License.
   7  *
   8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
   9  * or http://www.opensolaris.org/os/licensing.
  10  * See the License for the specific language governing permissions
  11  * and limitations under the License.
  12  *
  13  * When distributing Covered Code, include this CDDL HEADER in each
  14  * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  15  * If applicable, add the following below this CDDL HEADER, with the
  16  * fields enclosed by brackets "[]" replaced with your own identifying
  17  * information: Portions Copyright [yyyy] [name of copyright owner]
  18  *
  19  * CDDL HEADER END
  20  */

  22 #endif /* ! codereview */
  23 /*
  24  * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
  25  */

  27 #endif /* ! codereview */
  28 /*
  29  * Copyright 2005 Sun Microsystems, Inc.  All rights reserved.
  30  * Use is subject to license terms.
  31  */


  21 /* INDENT OFF */
  34 /*
  35  * __k_cos(double x; double y)
  36  * kernel cos function on [-pi/4, pi/4], pi/4 ~ 0.785398164
  37  * Input x is assumed to be bounded by ~pi/4 in magnitude.
  38  * Input y is the tail of x.
  39  *
  40  * Accurate Table look-up algorithm by K.C. Ng, May, 1995.
  41  *
  42  * Algorithm: see __sincos.c
  43  */

  45 #include "libm.h"

  47 static const double sc[] = {
  48 /* ONE  = */
  49         1.0,
  36 /* ONE  = */  1.0,
  50 /* NONE = */ -1.0,

  52 #endif /* ! codereview */
  53 /*
  54  * |sin(x) - (x+pp1*x^3+pp2*x^5)| <= 2^-58.79 for |x| < 0.008
  55  */
  56 /* PP1  = */-0.166666666666316558867252052378889521480627858683055567,
  57 /* PP2  = */.008333315652997472323564894248466758248475374977974017927,

  38 /* PP1  = */ -0.166666666666316558867252052378889521480627858683055567,
```



```
  39 /* PP2  = */   .008333315652997472323564894248466758248475374977974017927,
  59 /*
  60  * |(sin(x) - (x+p1*x^3+...+p4*x^9)|
  61  * |------------------------------ | <= 2^-57.63 for |x| < 0.1953125
  62  * |                x              |
  63  */
  64 /* P1   = */ -1.6666666666666296698052151389203015896 56e-0001,
  65 /* P2   = */ 8.333333332390951295683993455280336376663e-0003,
  66 /* P3   = */ -1.984126237997976692791551778230098403960e-0004,
  67 /* P4   = */ 2.753403624854277237649987622848330351110e-0006,

  69 #endif /* ! codereview */
  70 /*
  71  * |cos(x) - (1+qq1*x^2+qq2*x^4)| <= 2^-55.99 for |x| <= 0.008 (0x3f80624d)
  72  */
  73 /* QQ1  = */-0.499999999997549238184291198194841854274 2729,
  74 /* QQ2  = */0.041666542904352059294545209158357640398771740,

  49 /* QQ1  = */ -0.499999999997549238184291198194841854274 2729,
  50 /* QQ2  = */  0.041666542904352059294545209158357640398771740,
  76 /*
  77  * |cos(x) - (1+q1*x^2+...+q4*x^8)| <= 2^-55.86 for |x| <= 0.1640625 (10.5/64)
  78  */
  79 /* Q1   = */ -0.5,
  80 /* Q2   = */ 4.166666666500350703680945520860748617445e-0002,
  81 /* Q3   = */ -1.388888596436972210694266290577848696006e-0003,
  82 /* Q4   = */ 2.478563078858589473679519517892953492192e-0005, };

  57 /* Q4   = */  2.478563078858589473679519517892953492192e-0005,
  58 };
  59 /* INDENT ON */

  85 #define ONE             sc[0]
  86 #define NONE            sc[1]
  87 #define PP1             sc[2]
  88 #define PP2             sc[3]
  89 #define P1              sc[4]
  90 #define P2              sc[5]
  91 #define P3              sc[6]
  92 #define P4              sc[7]
  93 #define QQ1             sc[8]
  94 #define QQ2             sc[9]
  95 #define Q1              sc[10]
  96 #define Q2              sc[11]
  97 #define Q3              sc[12]
  98 #define Q4              sc[13]

 100 extern const double _TBL_sincos[], _TBL_sincosx[];

 102 double
 103 __k_cos(double x, double y)
 104 {
  79 __k_cos(double x, double y) {
 105         double z, w, s, v, p, q;
 106         int i, j, n, hx, ix;

 108         hx = ((int *)&x)[HIWORD];
 109         ix = hx & ~0x80000000;

 111         if (ix <= 0x3fc50000) {         /* |x| < 10.5/64 = 0.164062500 */
 112                 if (ix < 0x3e400000)    /* |x| < 2**-27 */
 113                         if ((int)x == 0)
 114                                 return (ONE);

 116 #endif /* ! codereview */
 117                 z = x * x;
```

```
#endif /* ! codereview */
                if (ix < 0x3f800000)    /* |x| < 0.008 */
                        q = z * (QQ1 + z * QQ2);
                else
                        q = z * ((Q1 + z * Q2) + (z * z) * (Q3 + z * Q4));

#endif /* ! codereview */
                return (ONE + q);
        } else {                        /* 0.164062500 < |x| < ~pi/4 */
                n = ix >> 20;
                i = (((ix >> 12) & 0xff) | 0x100) >> (0x401 - n);
                j = i - 10;

#endif /* ! codereview */
                if (hx < 0)
                        v = -y - (_TBL_sincosx[j] + x);
                else
                        v = y - (_TBL_sincosx[j] - x);

#endif /* ! codereview */
                s = v * v;
                j <<= 1;
                w = _TBL_sincos[j];
                z = _TBL_sincos[j + 1];
                z = _TBL_sincos[j+1];
                p = s * (PP1 + s * PP2);
                q = s * (QQ1 + s * QQ2);
                p = v + v * p;
                return (z - (w * p - z * q));
        }
}
```

_____*unchanged_portion_omitted_*

```
**********************************************************
    7457 Thu Jun  6 20:37:33 2019
new/usr/src/lib/libm/common/C/__lgamma.c
11210 libm should be cstyle(10NBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

#endif /* ! codereview */
/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2005 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

/*
 * double __k_lgamma(double x, int *signgamp);
 *
 * K.C. Ng, March, 1989.
 *
 * Part of the algorithm is based on W. Cody's lgamma function.
 */

#include "libm.h"

static const double one = 1.0,
        zero = 0.0,
        hln2pi = 0.91893853320467274178032973, /* log(2*pi)/2 */
        pi = 3.14159265358979323846264 33,
        two52 = 4503599627370496.0, /* 43300000,00000000 (used by sin_pi) */

static const double
one     = 1.0,
zero    = 0.0,
hln2pi  = 0.91893853320467274178032973,  /* log(2*pi)/2 */
pi      = 3.14159265358979323846264 33,
two52   = 4503599627370496.0,            /* 43300000,00000000 (used by sin_pi) */
/*
 * Numerator and denominator coefficients for rational minimax Approximation
 * P/Q over (0.5,1.5).
 */
        D1 = -5.772156649015328605195174e-1,
        p7 = 4.945235359296727046734888e0,
        p6 = 2.018112620856775083915565e2,
```



```
        p5 = 2.290838373831346393026739e3,
        p4 = 1.131967205903380828685045e4,
        p3 = 2.855724635671635335736389e4,
        p2 = 3.848496228443793359990269e4,
        p1 = 2.637748787624195437963534e4,
        p0 = 7.225813979700288197698961e3,
        q7 = 6.748212550303777196073036e1,
        q6 = 1.113332393857199323513008e3,
        q5 = 7.738757056935398733233834e3,
        q4 = 2.763987074403340708898585e4,
        q3 = 5.499310206226157329794414e4,
        q2 = 6.161122180066002127833352e4,
        q1 = 3.635127591501940507276287e4,
        q0 = 8.785536302431013170870835e3,

D1 =    -5.772156649015328605195174e-1,
p7 =     4.945235359296727046734888e0,
p6 =     2.018112620856775083915565e2,
p5 =     2.290838373831346393026739e3,
p4 =     1.131967205903380828685045e4,
p3 =     2.855724635671635335736389e4,
p2 =     3.848496228443793359990269e4,
p1 =     2.637748787624195437963534e4,
p0 =     7.225813979700288197698961e3,
q7 =     6.748212550303777196073036e1,
q6 =     1.113332393857199323513008e3,
q5 =     7.738757056935398733233834e3,
q4 =     2.763987074403340708898585e4,
q3 =     5.499310206226157329794414e4,
q2 =     6.161122180066002127833352e4,
q1 =     3.635127591501940507276287e4,
q0 =     8.785536302431013170870835e3,
/*
 * Numerator and denominator coefficients for rational minimax Approximation
 * G/H over (1.5,4.0).
 */
        D2 = 4.227843350984671393993777e-1,
        g7 = 4.974607845568932035012064e0,
        g6 = 5.424138599891070494101986e2,
        g5 = 1.550693864978364947665077e4,
        g4 = 1.847932904445632425417223e5,
        g3 = 1.088204769468828767498470e6,
        g2 = 3.338152967987029735917223e6,
        g1 = 5.106661678927352456275255e6,
        g0 = 3.074109054850539556250927e6,
        h7 = 1.830328399370592604055942e2,
        h6 = 7.765049321445005871323047e3,
        h5 = 1.331903827966074194402448e5,
        h4 = 1.136705821321969608938755e6,
        h3 = 5.267964117437946917577538e6,
        h2 = 1.346701454311101692290052e7,
        h1 = 1.782736530353274213975932e7,
        h0 = 9.533095591844353613395747e6,

D2 =     4.227843350984671393993777e-1,
g7 =     4.974607845568932035012064e0,
g6 =     5.424138599891070494101986e2,
g5 =     1.550693864978364947665077e4,
g4 =     1.847932904445632425417223e5,
g3 =     1.088204769468828767498470e6,
g2 =     3.338152967987029735917223e6,
g1 =     5.106661678927352456275255e6,
g0 =     3.074109054850539556250927e6,
h7 =     1.830328399370592604055942e2,
h6 =     7.765049321445005871323047e3,
h5 =     1.331903827966074194402448e5,
```

```
 64 h4 =     1.136705821321969608938755e6,
 65 h3 =     5.267964117437946917577538e6,
 66 h2 =     1.346701454311101692290052e7,
 67 h1 =     1.782736530353274213975932e7,
 68 h0 =     9.533095591844353613395747e6,
 93 /*
 94  * Numerator and denominator coefficients for rational minimax Approximation
 95  * U/V over (4.0,12.0).
 96  */
 97         D4 = 1.791759469228055000094023e0,
 98         u7 = 1.474502166059939948905062e4,
 99         u6 = 2.426813369486704502836312e6,
100         u5 = 1.214755574045093227939592e8,
101         u4 = 2.663432449630976949898078e9,
102         u3 = 2.940378956634553899906876e10,
103         u2 = 1.702665737765398868392998e11,
104         u1 = 4.926125793377430887588120e11,
105         u0 = 5.606251856223951465078242e11,
106         v7 = 2.690530175870899333379843e3,
107         v6 = 6.393885654300092398984238e5,
108         v5 = 4.135599930241388052042842e7,
109         v4 = 1.120872109616147941376570e9,
110         v3 = 1.488613728678813811542398e10,
111         v2 = 1.016803586272438228077304e11,
112         v1 = 3.417476345507377132798597e11,
113         v0 = 4.463158187419713286462081e11,

 73 D4 =     1.791759469228055000094023e0,
 74 u7 =     1.474502166059939948905062e4,
 75 u6 =     2.426813369486704502836312e6,
 76 u5 =     1.214755574045093227939592e8,
 77 u4 =     2.663432449630976949898078e9,
 78 u3 =     2.940378956634553899906876e10,
 79 u2 =     1.702665737765398868392998e11,
 80 u1 =     4.926125793377430887588120e11,
 81 u0 =     5.606251856223951465078242e11,
 82 v7 =     2.690530175870899333379843e3,
 83 v6 =     6.393885654300092398984238e5,
 84 v5 =     4.135599930241388052042842e7,
 85 v4 =     1.120872109616147941376570e9,
 86 v3 =     1.488613728678813811542398e10,
 87 v2 =     1.016803586272438228077304e11,
 88 v1 =     3.417476345507377132798597e11,
 89 v0 =     4.463158187419713286462081e11,
115 /*
116  * Coefficients for minimax approximation over (12, INF).
117  */
118         c5 = -1.910444077728e-03,
119         c4 = 8.4171387781295e-04,
120         c3 = -5.952379913043012e-04,
121         c2 = 7.93650793500350248e-04,
122         c1 = -2.777777777777681622553e-03,
123         c0 = 8.333333333333333331554247e-02,
124         c6 = 5.7083835261e-03;
 93 c5 =    -1.910444077728e-03,
 94 c4 =     8.4171387781295e-04,
 95 c3 =    -5.952379913043012e-04,
 96 c2 =     7.93650793500350248e-04,
 97 c1 =    -2.777777777777681622553e-03,
 98 c0 =     8.333333333333333331554247e-02,
 99 c6 =     5.7083835261e-03;

126 /*
127  * Return sin(pi*x).  We assume x is finite and negative, and if it
128  * is an integer, then the sign of the zero returned doesn't matter.
129  */
```



```
130 static double
131 sin_pi(double x)
132 {
106 sin_pi(double x) {
133         double y, z;
134         int n;

136         y = -x;

138 #endif /* ! codereview */
139         if (y <= 0.25)
140                 return (__k_sin(pi * x, 0.0));

142 #endif /* ! codereview */
143         if (y >= two52)
144                 return (zero);

146 #endif /* ! codereview */
147         z = floor(y);

149 #endif /* ! codereview */
150         if (y == z)
151                 return (zero);

153         /* argument reduction: set y = |x| mod 2 */
154         y *= 0.5;
155         y = 2.0 * (y - floor(y));

157         /* now floor(y * 4) tells which octant y is in */
158         n = (int)(y * 4.0);

160 #endif /* ! codereview */
161         switch (n) {
162         case 0:
163                 y = __k_sin(pi * y, 0.0);
164                 break;
165         case 1:
166         case 2:
167                 y = __k_cos(pi * (0.5 - y), 0.0);
168                 break;
169         case 3:
170         case 4:
171                 y = __k_sin(pi * (1.0 - y), 0.0);
172                 break;
173         case 5:
174         case 6:
175                 y = -__k_cos(pi * (y - 1.5), 0.0);
176                 break;
177         default:
178                 y = __k_sin(pi * (y - 2.0), 0.0);
179                 break;
180         }

182 #endif /* ! codereview */
183         return (-y);
184 }

186 static double
187 neg(double z, int *signgamp)
188 {
111 neg(double z, int *signgamp) {
189         double t, p;

191         /* BEGIN CSTYLED */
192 #endif /* ! codereview */
193         /*
```

```
194          * written by K.C. Ng,  Feb 2, 1989.
195          *
196          * Since
197          *              -z*G(-z)*G(z) = pi/sin(pi*z),
198          * we have
199          *      G(-z) = -pi/(sin(pi*z)*G(z)*z)
200          *            =  pi/(sin(pi*(-z))*G(z)*z)
201          * Algorithm
202          *              z = |z|
203          *              t = sin_pi(z); ...note that when z>2**52, z is an int
204          *              and hence t=0.
205          *
206          *              if (t == 0.0) return 1.0/0.0;
207          *              if (t< 0.0) *signgamp = -1; else t= -t;
208          *              if (z+1.0 == 1.0)       ...tiny z
209          *                  return -log(z);
210          *              else
211          *                  return log(pi/(t*z))-__k_lgamma(z, signgamp);
212          */
213         /* END CSTYLED */
214 #endif /* ! codereview */

216         t = sin_pi(z);                  /* t := sin(pi*z) */

218 #endif /* ! codereview */
219         if (t == zero)                  /* return 1.0/0.0 = +INF */
220                 return (one / fabs(t));

222 #endif /* ! codereview */
223         z = -z;
224         p = z + one;

226 #endif /* ! codereview */
227         if (p == one)
228                 p = -log(z);
229         else
230                 p = log(pi / (fabs(t) * z)) - __k_lgamma(z, signgamp);

232 #endif /* ! codereview */
233         if (t < zero)
234                 *signgamp = -1;

236 #endif /* ! codereview */
237         return (p);
238 }

240 double
241 __k_lgamma(double x, int *signgamp)
242 {
114 __k_lgamma(double x, int *signgamp) {
243         double t, p, q, cr, y;

245         /* purge off +-inf, NaN and negative arguments */
246         if (!finite(x))
247                 return (x * x);

249 #endif /* ! codereview */
250         *signgamp = 1;

252 #endif /* ! codereview */
253         if (signbit(x))
254                 return (neg(x, signgamp));

256         /* lgamma(x) ~ log(1/x) for really tiny x */
257         t = one + x;
```



```
259 #endif /* ! codereview */
260         if (t == one) {
261                 if (x == zero)
262                         return (one / x);

264 #endif /* ! codereview */
265                 return (-log(x));
266         }

268         /* for tiny < x < inf */
269         if (x <= 1.5) {
270                 if (x < 0.6796875) {
271                         cr = -log(x);
272                         y = x;
273                 } else {
274                         cr = zero;
275                         y = x - one;
276                 }

278                 if (x <= 0.5 || x >= 0.6796875) {
279                         if (x == one)
280                                 return (zero);

282                         p = p0 + y * (p1 + y * (p2 + y * (p3 + y * (p4 + y *
283                             (p5 + y * (p6 + y * p7))))));
284                         q = q0 + y * (q1 + y * (q2 + y * (q3 + y * (q4 + y *
285                             (q5 + y * (q6 + y * (q7 + y)))))));
286                         return (cr + y * (D1 + y * (p / q)));
120                         p = p0+y*(p1+y*(p2+y*(p3+y*(p4+y*(p5+y*(p6+y*p7))))));
121                         q = q0+y*(q1+y*(q2+y*(q3+y*(q4+y*(q5+y*(q6+y*
122                             (q7+y)))))));
123                         return (cr+y*(D1+y*(p/q)));
287                 } else {
288                         y = x - one;
289                         p = g0 + y * (g1 + y * (g2 + y * (g3 + y * (g4 + y *
290                             (g5 + y * (g6 + y * g7))))));
291                         q = h0 + y * (h1 + y * (h2 + y * (h3 + y * (h4 + y *
292                             (h5 + y * (h6 + y * (h7 + y)))))));
293                         return (cr + y * (D2 + y * (p / q)));
126                         p = g0+y*(g1+y*(g2+y*(g3+y*(g4+y*(g5+y*(g6+y*g7))))));
127                         q = h0+y*(h1+y*(h2+y*(h3+y*(h4+y*(h5+y*(h6+y*
128                             (h7+y)))))));
129                         return (cr+y*(D2+y*(p/q)));
294                 }
295         } else if (x <= 4.0) {
296                 if (x == 2.0)
297                         return (zero);

299 #endif /* ! codereview */
300                 y = x - 2.0;
301                 p = g0 + y * (g1 + y * (g2 + y * (g3 + y * (g4 + y * (g5 + y *
302                     (g6 + y * g7))))));
303                 q = h0 + y * (h1 + y * (h2 + y * (h3 + y * (h4 + y * (h5 + y *
304                     (h6 + y * (h7 + y)))))));
305                 return (y * (D2 + y * (p / q)));
134                 p = g0+y*(g1+y*(g2+y*(g3+y*(g4+y*(g5+y*(g6+y*g7))))));
135                 q = h0+y*(h1+y*(h2+y*(h3+y*(h4+y*(h5+y*(h6+y*(h7+y)))))));
136                 return (y*(D2+y*(p/q)));
306         } else if (x <= 12.0) {
307                 y = x - 4.0;
308                 p = u0 + y * (u1 + y * (u2 + y * (u3 + y * (u4 + y * (u5 + y *
309                     (u6 + y * u7))))));
310                 q = v0 + y * (v1 + y * (v2 + y * (v3 + y * (v4 + y * (v5 + y *
311                     (v6 + y * (v7 - y)))))));
312                 return (D4 + y * (p / q));
139                 p = u0+y*(u1+y*(u2+y*(u3+y*(u4+y*(u5+y*(u6+y*u7))))));
```

```
                q = v0+y*(v1+y*(v2+y*(v3+y*(v4+y*(v5+y*(v6+y*(v7-y)))))));
                return (D4+y*(p/q));
        } else if (x <= 1.0e17) {       /* x ~< 2**(prec+3) */
                t = one / x;
                y = t * t;
                p = hln2pi + t * (c0 + y * (c1 + y * (c2 + y * (c3 + y * (c4 +
                    y * (c5 + y * c6))))));
                p = hln2pi+t*(c0+y*(c1+y*(c2+y*(c3+y*(c4+y*(c5+y*c6))))));
                q = log(x);
                return (x * (q - one) - (0.5 * q - p));
                return (x*(q-one)-(0.5*q-p));
        } else {                        /* may overflow */
                return (x * (log(x) - 1.0));
        }
}
```

_____*unchanged_portion_omitted_*

```
**********************************************************
    1078 Thu Jun  6 20:37:33 2019
new/usr/src/lib/libm/common/C/__libx_errno.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

#endif /* ! codereview */
/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

extern int *___errno(void);

int *
__libm_errno(void)
{
__libm_errno(void) {
        return (___errno());
}
______unchanged_portion_omitted_
```

```
**********************************************************
    4322 Thu Jun  6 20:37:34 2019
new/usr/src/lib/libm/common/C/__rem_pio2.c
11210 libm should be cstyle(10NBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

#endif /* ! codereview */
/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2005 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

/*
 * __rem_pio2(x, y) passes back a better-than-double-precision
 * approximation to x mod pi/2 in y[0]+y[1] and returns an integer
 * congruent mod 8 to the integer part of x/(pi/2).
 *
 * This implementation tacitly assumes that x is finite and at
 * least about pi/4 in magnitude.
 */

#include "libm.h"

extern const int _TBL_ipio2_inf[];


/* INDENT OFF */
/*
 * invpio2:  53 bits of 2/pi
 * pio2_1:   first  33 bit of pi/2
 * pio2_1t:  pi/2 - pio2_1
 * pio2_2:   second 33 bit of pi/2
 * pio2_2t:  pi/2 - pio2_2
 * pio2_3:   third  33 bit of pi/2
 * pio2_3t:  pi/2 - pio2_3
 */
static const double half = 0.5,
static const double
        half    = 0.5,
        invpio2 = 0.63661977236758134307553 5,   /* 2^ -1  * 1.45F306DC9C883 */
        pio2_1 = 1.57079632673412561416 6,       /* 2^  0  * 1.921FB54400000 */
```



```
        pio2_1t = 6.077100506506192601475e-11,  /* 2^-34  * 1.0B4611A626331 */
        pio2_2 = 6.077100506303965976596e-11,   /* 2^-34  * 1.0B4611A600000 */
        pio2_2t = 2.022266248795950732400e-21,  /* 2^-69  * 1.3198A2E037073 */
        pio2_3 = 2.022266248711166455796e-21,   /* 2^-69  * 1.3198A2E000000 */
        pio2_3t = 8.478427660368899643959e-32;  /* 2^-104 * 1.B839A252049C1 */

/* INDENT ON */

int
__rem_pio2(double x, double *y)
{
__rem_pio2(double x, double *y) {
        double w, t, r, fn;
        double tx[3];
        int e0, i, j, nx, n, ix, hx, lx;

        hx = ((int *)&x)[HIWORD];
        ix = hx & 0x7fffffff;

        if (ix < 0x4002d97c) {
                /* |x| < 3pi/4, special case with n=1 */
                t = fabs(x) - pio2_1;

#endif /* ! codereview */
                if (ix != 0x3ff921fb) { /* 33+53 bit pi is good enough */
                        y[0] = t - pio2_1t;
                        y[1] = (t - y[0]) - pio2_1t;
                } else {                /* near pi/2, use 33+33+53 bit pi */
                        t -= pio2_2;
                        y[0] = t - pio2_2t;
                        y[1] = (t - y[0]) - pio2_2t;
                }

#endif /* ! codereview */
                if (hx < 0) {
                        y[0] = -y[0];
                        y[1] = -y[1];
                        return (-1);
                }

#endif /* ! codereview */
                return (1);
        }

        if (ix <= 0x413921fb) {
                /* |x| <= 2^19 pi */
                t = fabs(x);
                n = (int)(t * invpio2 + half);
                fn = (double)n;
                r = t - fn * pio2_1;
                j = ix >> 20;
                w = fn * pio2_1t;       /* 1st round good to 85 bit */
                y[0] = r - w;
                i = j - ((((int *)y)[HIWORD] >> 20) & 0x7ff);

#endif /* ! codereview */
                if (i > 16) {           /* 2nd iteration (rare) */
                        /* 2nd round good to 118 bit */
                        if (i < 35) {
                                t = r;  /* r-fn*pio2_2 may not be exact */
                                w = fn * pio2_2;
                                r = t - w;
                                w = fn * pio2_2t - ((t - r) - w);
                                y[0] = r - w;
                        } else {
                                r -= fn * pio2_2;
```

```
                                w = fn * pio2_2t;
                                y[0] = r - w;
                                i = j - ((((int *)y)[HIWORD] >> 20) & 0x7ff);

#endif /* ! codereview */
                                if (i > 49) {
                                        /* 3rd iteration (extremely rare) */
                                        if (i < 68) {
                                                t = r;
                                                w = fn * pio2_3;
                                                r = t - w;
                                                w = fn * pio2_3t - ((t - r) -
                                                    w);
                                                w = fn * pio2_3t -
                                                    ((t - r) - w);
                                                y[0] = r - w;
                                        } else {
                                                /*
                                                 * 3rd round good to 151 bits;
                                                 * covered all possible cases
                                                 */
                                                r -= fn * pio2_3;
                                                w = fn * pio2_3t;
                                                y[0] = r - w;
                                        }
                                }
                        }
                }

#endif /* ! codereview */
                y[1] = (r - y[0]) - w;

#endif /* ! codereview */
                if (hx < 0) {
                        y[0] = -y[0];
                        y[1] = -y[1];
                        return (-n);
                }

#endif /* ! codereview */
                return (n);
        }

        e0 = (ix >> 20) - 1046;         /* e0 = ilogb(x)-23; */

        /* break x into three 24 bit pieces */
        lx = ((int *)&x)[LOWORD];
        i = (lx & 0x1f) << 19;
        tx[2] = (double)i;
        j = (lx >> 5) & 0xffffff;
        tx[1] = (double)j;
        tx[0] = (double)((((ix & 0xfffff) | 0x100000) << 3) | ((unsigned)lx >>
            29));
        tx[0] = (double)((((ix & 0xfffff) | 0x100000) << 3) |
            ((unsigned)lx >> 29));
        nx = 3;

#endif /* ! codereview */
        if (i == 0) {
                /* skip zero term */
                nx--;

#endif /* ! codereview */
                if (j == 0)
                        nx--;
        }
```



```
#endif /* ! codereview */
        n = __rem_pio2m(tx, y, e0, nx, 2, _TBL_ipio2_inf);

#endif /* ! codereview */
        if (hx < 0) {
                y[0] = -y[0];
                y[1] = -y[1];
                return (-n);
        }

#endif /* ! codereview */
        return (n);
}
```

```
**********************************************************
    8780 Thu Jun  6 20:37:34 2019
new/usr/src/lib/libm/common/C/__rem_pio2m.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

#endif /* ! codereview */
/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2005 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

/*
 * int __rem_pio2m(x,y,e0,nx,prec,ipio2)
 * double x[],y[]; int e0,nx,prec; const int ipio2[];
 *
 * __rem_pio2m return the last three digits of N with
 *              y = x - N*pi/2
 * so that |y| < pi/4.
 *
 * The method is to compute the integer (mod 8) and fraction parts of
 * (2/pi)*x without doing the full multiplication. In general we
 * skip the part of the product that are known to be a huge integer (
 * more accurately, = 0 mod 8 ). Thus the number of operations are
 * independent of the exponent of the input.
 *
 * (2/PI) is represented by an array of 24-bit integers in ipio2[].
 * Here PI could as well be a machine value pi.
 *
 * Input parameters:
 *      x[]     The input value (must be positive) is broken into nx
 *              pieces of 24-bit integers in double precision format.
 *              x[i] will be the i-th 24 bit of x. The scaled exponent
 *              of x[0] is given in input parameter e0 (i.e., x[0]*2^e0
 *              match x’s up to 24 bits.
 *
 *              Example of breaking a double z into x[0]+x[1]+x[2]:
 *                      e0 = ilogb(z)-23
 *                      z  = scalbn(z,-e0)
 *              for i = 0,1,2
 *                      x[i] =  floor(z)
```



```
 *                      z    = (z-x[i])*2**24
 *
 *
 *      y[]     ouput result in an array of double precision numbers.
 *              The dimension of y[] is:
 *                      24-bit  precision       1
 *                      53-bit  precision       2
 *                      64-bit  precision       2
 *                      113-bit precision       3
 *              The actual value is the sum of them. Thus for 113-bit
 *              precsion, one may have to do something like:
 *
 *              long double t,w,r_head, r_tail;
 *              t = (long double)y[2] + (long double)y[1];
 *              w = (long double)y[0];
 *              r_head = t+w;
 *              r_tail = w - (r_head - t);
 *
 *      e0      The exponent of x[0]
 *
 *      nx      dimension of x[]
 *
 *      prec    an interger indicating the precision:
 *                      0       24  bits (single)
 *                      1       53  bits (double)
 *                      2       64  bits (extended)
 *                      3       113 bits (quad)
 *
 *      ipio2[]
 *              integer array, contains the (24*i)-th to (24*i+23)-th
 *              bit of 2/pi or 2/PI after binary point. The corresponding
 *              floating value is
 *
 *                      ipio2[i] * 2^(-24(i+1)).
 *
 * External function:
 *      double scalbn( ), floor( );
 *
 *
 * Here is the description of some local variables:
 *
 *      jk      jk+1 is the initial number of terms of ipio2[] needed
 *              in the computation. The recommended value is 3,4,4,
 *              6 for single, double, extended,and quad.
 *
 *      jz      local integer variable indicating the number of
 *              terms of ipio2[] used.
 *
 *      jx      nx - 1
 *
 *      jv      index for pointing to the suitable ipio2[] for the
 *              computation. In general, we want
 *                      ( 2^e0*x[0] * ipio2[jv-1]*2^(-24jv) )/8
 *              is an integer. Thus
 *                      e0-3-24*jv >= 0 or (e0-3)/24 >= jv
 *              Hence jv = max(0,(e0-3)/24).
 *
 *      jp      jp+1 is the number of terms in pio2[] needed, jp = jk.
 *
 *      q[]     double array with integral value, representing the
 *              24-bits chunk of the product of x and 2/pi.
 *
 *      q0      the corresponding exponent of q[0]. Note that the
 *              exponent for q[i] would be q0-24*i.
 *
 *      pio2[]  double precision array, obtained by cutting pi/2
```

```
 *               into 24 bits chunks.
 *
 *      f[]      ipio2[] in floating point
 *
 *      iq[]     integer array by breaking up q[] in 24-bits chunk.
 *
 *      fq[]     final product of x*(2/pi) in fq[0],..,fq[jk]
 *
 *      ih       integer. If >0 it indicats q[] is >= 0.5, hence
 *               it also indicates the *sign* of the result.
 *
 */

#include "libm.h"

#if defined(__i386) && !defined(__amd64)
extern int __swapRP(int);
#endif

static const int init_jk[] = { 3, 4, 4, 6 };    /* initial value for jk */

static const double pio2[] = {
        1.57079625129699707031e+00, 7.54978941586159635335e-08,
        5.39030252995776476554e-15, 3.28200341580791294123e-22,
        1.27065575308067607349e-29, 1.22933308981111328932e-36,
        2.73370053816464559624e-44, 2.16741683877804819444e-51,
        1.57079625129699707031e+00,
        7.54978941586159635335e-08,
        5.39030252995776476554e-15,
        3.28200341580791294123e-22,
        1.27065575308067607349e-29,
        1.22933308981111328932e-36,
        2.73370053816464559624e-44,
        2.16741683877804819444e-51,
};

static const double zero = 0.0,
static const double
        zero    = 0.0,
        one = 1.0,
        half = 0.5,
        eight = 8.0,
        eighth = 0.125,
        two24 = 16777216.0,
        twon24 = 5.96046447753906250E-8;

int
__rem_pio2m(double *x, double *y, int e0, int nx, int prec, const int *ipio2)
{
        int jz, jx, jv, jp, jk, carry, n, iq[20];
        int i, j, k, m, q0, ih;
        double z, fw, f[20], fq[20], q[20];

#endif /* ! codereview */
#if defined(__i386) && !defined(__amd64)
        int rp;

        rp = __swapRP(fp_extended);
#endif

        /* initialize jk */
        jp = jk = init_jk[prec];

        /* determine jx,jv,q0, note that 3>q0 */
        jx = nx - 1;
        jv = (e0 - 3) / 24;
```



```
#endif /* ! codereview */
        if (jv < 0)
                jv = 0;

#endif /* ! codereview */
        q0 = e0 - 24 * (jv + 1);

        /* set up f[0] to f[jx+jk] where f[jx+jk] = ipio2[jv+jk] */
        j = jv - jx;
        m = jx + jk;

#endif /* ! codereview */
        for (i = 0; i <= m; i++, j++)
                f[i] = (j < 0) ? zero : (double)ipio2[j];
                f[i] = (j < 0)? zero : (double)ipio2[j];

        /* compute q[0],q[1],...q[jk] */
        for (i = 0; i <= jk; i++) {
                for (j = 0, fw = zero; j <= jx; j++)
                        fw += x[j] * f[jx + i - j];

                        fw += x[j] * f[jx+i-j];
                q[i] = fw;
        }

        jz = jk;

#endif /* ! codereview */
recompute:
        /* distill q[] into iq[] reversingly */
        for (i = 0, j = jz, z = q[jz]; j > 0; i++, j--) {
                fw = (double)((int)(twon24 * z));
                iq[i] = (int)(z - two24 * fw);
                z = q[j - 1] + fw;
                z = q[j-1] + fw;
        }

        /* compute n */
        z = scalbn(z, q0);              /* actual value of z */
        z -= eight * floor(z * eighth); /* trim off integer >= 8 */
        n = (int)z;
        z -= (double)n;
        ih = 0;

#endif /* ! codereview */
        if (q0 > 0) {                   /* need iq[jz-1] to determine n */
                i = (iq[jz - 1] >> (24 - q0));
                i = (iq[jz-1] >> (24 - q0));
                n += i;
                iq[jz - 1] -= i << (24 - q0);
                ih = iq[jz - 1] >> (23 - q0);
                iq[jz-1] -= i << (24 - q0);
                ih = iq[jz-1] >> (23 - q0);
        } else if (q0 == 0) {
                ih = iq[jz - 1] >> 23;
                ih = iq[jz-1] >> 23;
        } else if (z >= half) {
                ih = 2;
        }

        if (ih > 0) {                   /* q > 0.5 */
                n += 1;
                carry = 0;

#endif /* ! codereview */
```

```
                for (i = 0; i < jz; i++) {      /* compute 1-q */
                        j = iq[i];

#endif /* ! codereview */
                        if (carry == 0) {
                                if (j != 0) {
                                        carry = 1;
                                        iq[i] = 0x1000000 - j;
                                }
                        } else {
                                iq[i] = 0xffffff - j;
                        }
                }

#endif /* ! codereview */
                if (q0 > 0) {           /* rare case: chance is 1 in 12 */
                        switch (q0) {
                        case 1:
                                iq[jz - 1] &= 0x7fffff;
                                iq[jz-1] &= 0x7fffff;
                                break;
                        case 2:
                                iq[jz - 1] &= 0x3fffff;
                                iq[jz-1] &= 0x3fffff;
                                break;
                        }
                }

#endif /* ! codereview */
                if (ih == 2) {
                        z = one - z;

#endif /* ! codereview */
                        if (carry != 0)
                                z -= scalbn(one, q0);
                }
        }

        /* check if recomputation is needed */
        if (z == zero) {
                j = 0;

#endif /* ! codereview */
                for (i = jz - 1; i >= jk; i--)
                        j |= iq[i];

#endif /* ! codereview */
                if (j == 0) {           /* need recomputation */
                        /* set k to no. of terms needed */
                        for (k = 1; iq[jk - k] == 0; k++)
                        for (k = 1; iq[jk-k] == 0; k++)
                                ;

                        /* add q[jz+1] to q[jz+k] */
                        for (i = jz + 1; i <= jz + k; i++) {
                                f[jx + i] = (double)ipio2[jv + i];

                                f[jx+i] = (double)ipio2[jv+i];
                                for (j = 0, fw = zero; j <= jx; j++)
                                        fw += x[j] * f[jx + i - j];

                                        fw += x[j] * f[jx+i-j];
                                q[i] = fw;
                        }

#endif /* ! codereview */
```



```
                        jz += k;
                        goto recompute;
                }
        }

        /* cut out zero terms */
        if (z == zero) {
                jz -= 1;
                q0 -= 24;

#endif /* ! codereview */
                while (iq[jz] == 0) {
                        jz--;
                        q0 -= 24;
                }
        } else {                        /* break z into 24-bit if neccessary */
                z = scalbn(z, -q0);

#endif /* ! codereview */
                if (z >= two24) {
                        fw = (double)((int)(twon24 * z));
                        iq[jz] = (int)(z - two24 * fw);
                        jz += 1;
                        q0 += 24;
                        iq[jz] = (int)fw;
                } else {
                        iq[jz] = (int)z;
                }
        }

        /* convert integer "bit" chunk to floating-point value */
        fw = scalbn(one, q0);

#endif /* ! codereview */
        for (i = jz; i >= 0; i--) {
                q[i] = fw * (double)iq[i];
                fw *= twon24;
        }

        /* compute pio2[0,...,jp]*q[jz,...,0] */
        for (i = jz; i >= 0; i--) {
                for (fw = zero, k = 0; k <= jp && k <= jz - i; k++)
                        fw += pio2[k] * q[i + k];

                fq[jz - i] = fw;
                        fw += pio2[k] * q[i+k];
                fq[jz-i] = fw;
        }

        /* compress fq[] into y[] */
        switch (prec) {
        case 0:
                fw = zero;

#endif /* ! codereview */
                for (i = jz; i >= 0; i--)
                        fw += fq[i];

                y[0] = (ih == 0) ? fw : -fw;
                y[0] = (ih == 0)? fw : -fw;
                break;

        case 1:
        case 2:
                fw = zero;
```

```
366 #endif /* ! codereview */
367                 for (i = jz; i >= 0; i--)
368                         fw += fq[i];

370                 y[0] = (ih == 0) ? fw : -fw;
111                 y[0] = (ih == 0)? fw : -fw;
371                 fw = fq[0] - fw;

373 #endif /* ! codereview */
374                 for (i = 1; i <= jz; i++)
375                         fw += fq[i];

377                 y[1] = (ih == 0) ? fw : -fw;
113                 y[1] = (ih == 0)? fw : -fw;
378                 break;

380         default:

382 #endif /* ! codereview */
383                 for (i = jz; i > 0; i--) {
384                         fw = fq[i - 1] + fq[i];
385                         fq[i] += fq[i - 1] - fw;
386                         fq[i - 1] = fw;
117                         fw = fq[i-1] + fq[i];
118                         fq[i] += fq[i-1] - fw;
119                         fq[i-1] = fw;
387                 }

389 #endif /* ! codereview */
390                 for (i = jz; i > 1; i--) {
391                         fw = fq[i - 1] + fq[i];
392                         fq[i] += fq[i - 1] - fw;
393                         fq[i - 1] = fw;
121                         fw = fq[i-1] + fq[i];
122                         fq[i] += fq[i-1] - fw;
123                         fq[i-1] = fw;
394                 }

396 #endif /* ! codereview */
397                 for (fw = zero, i = jz; i >= 2; i--)
398                         fw += fq[i];

400 #endif /* ! codereview */
401                 if (ih == 0) {
402                         y[0] = fq[0];
403                         y[1] = fq[1];
404                         y[2] = fw;
405                 } else {
406                         y[0] = -fq[0];
407                         y[1] = -fq[1];
408                         y[2] = -fw;
409                 }
410         }

412 #if defined(__i386) && !defined(__amd64)
413         (void) __swapRP(rp);
414 #endif
415         return (n & 7);
416 }
```

```
**********************************************************
    3697 Thu Jun  6 20:37:35 2019
new/usr/src/lib/libm/common/C/__sin.c
11210 libm should be cstyle(10NBLD) clean
**********************************************************
   1 /*
   2  * CDDL HEADER START
   3  *
   4  * The contents of this file are subject to the terms of the
   5  * Common Development and Distribution License (the "License").
   6  * You may not use this file except in compliance with the License.
   7  *
   8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
   9  * or http://www.opensolaris.org/os/licensing.
  10  * See the License for the specific language governing permissions
  11  * and limitations under the License.
  12  *
  13  * When distributing Covered Code, include this CDDL HEADER in each
  14  * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  15  * If applicable, add the following below this CDDL HEADER, with the
  16  * fields enclosed by brackets "[]" replaced with your own identifying
  17  * information: Portions Copyright [yyyy] [name of copyright owner]
  18  *
  19  * CDDL HEADER END
  20  */

  22 #endif /* ! codereview */
  23 /*
  24  * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
  25  */

  27 #endif /* ! codereview */
  28 /*
  29  * Copyright 2005 Sun Microsystems, Inc.  All rights reserved.
  30  * Use is subject to license terms.
  31  */


  21 /* INDENT OFF */
  34 /*
  35  * __k_sin( double x;  double y )
  36  * kernel sin function on [-pi/4, pi/4], pi/4 ~ 0.785398164
  37  * Input x is assumed to be bounded by ~pi/4 in magnitude.
  38  * Input y is the tail of x.
  39  *
  40  * Accurate Table look-up algorithm by K.C. Ng, May, 1995.
  41  *
  42  * Algorithm: see __sincos.c
  43  */

  45 #include "libm.h"

  47 static const double sc[] = {
  48 /* ONE  = */
  49         1.0,
  36 /* ONE  = */  1.0,
  50 /* NONE = */ -1.0,

  52 #endif /* ! codereview */
  53 /*
  54  * |sin(x) - (x+pp1*x^3+pp2*x^5)| <= 2^-58.79 for |x| < 0.008
  55  */
  56 /* PP1  = */-0.16666666666631655886725205237888952148062785868305556 7,
  57 /* PP2  = */.00833331565299747232356489424846675824847537497797401 7927,

  38 /* PP1  = */ -0.16666666666631655886725205237888952148062785868305556 7,
```



```
  39 /* PP2  = */   .00833331565299747232356489424846675824847537497797401 7927,
  59 /*
  60  * |(sin(x) - (x+p1*x^3+...+p4*x^9)|
  61  * |------------------------------ | <= 2^-57.63 for |x| < 0.1953125
  62  * |                x              |
  63  */
  64 /* P1   = */ -1.66666666666662966980521513892030158965 6e-0001,
  65 /* P2   = */ 8.33333333239095129568399345528033637666 3e-0003,
  66 /* P3   = */ -1.98412623799797669279155177823009840396 0e-0004,
  67 /* P4   = */ 2.75340362485427723764998762284833035111 0e-0006,

  69 #endif /* ! codereview */
  70 /*
  71  * |cos(x) - (1+qq1*x^2+qq2*x^4)| <= 2^-55.99 for |x| <= 0.008 (0x3f80624d)
  72  */
  73 /* QQ1  = */-0.49999999999754923818429119819484185427427 29,
  74 /* QQ2  = */0.04166654290435205929454520915835764039877174 0,

  49 /* QQ1  = */ -0.49999999999754923818429119819484185427427 29,
  50 /* QQ2  = */  0.04166654290435205929454520915835764039877174 0,
  76 /*
  77  * |cos(x) - (1+q1*x^2+...+q4*x^8)| <= 2^-55.86 for |x| <= 0.1640625 (10.5/64)
  78  */
  79 /* Q1   = */ -0.5,
  80 /* Q2   = */ 4.16666666650035070368094552086074861744 5e-0002,
  81 /* Q3   = */ -1.38888859643697221069426629057784869600 6e-0003,
  82 /* Q4   = */ 2.47856307885858947367951951789295349219 2e-0005,
  83 };

  59 /* INDENT ON */

  86 #define ONE             sc[0]
  87 #define NONE            sc[1]
  88 #define PP1             sc[2]
  89 #define PP2             sc[3]
  90 #define P1              sc[4]
  91 #define P2              sc[5]
  92 #define P3              sc[6]
  93 #define P4              sc[7]
  94 #define QQ1             sc[8]
  95 #define QQ2             sc[9]
  96 #define Q1              sc[10]
  97 #define Q2              sc[11]
  98 #define Q3              sc[12]
  99 #define Q4              sc[13]

 101 extern const double _TBL_sincos[], _TBL_sincosx[];

 103 double
 104 __k_sin(double x, double y)
 105 {
  79 __k_sin(double x, double y) {
 106         double z, w, s, v, p, q;
 107         int i, j, n, hx, ix;

 109         hx = ((int *)&x)[HIWORD];
 110         ix = hx & ~0x80000000;

 112         if (ix <= 0x3fc50000) {         /* |x| < 10.5/64 = 0.164062500 */
 113                 if (ix < 0x3e400000)    /* |x| < 2**-27 */
 114                         if ((int)x == 0)
 115                                 return (x + y);

 117 #endif /* ! codereview */
 118                 z = x * x;
```

```
#endif /* ! codereview */
                if (ix < 0x3f800000)    /* |x| < 0.008  */
                        p = (x * z) * (PP1 + z * PP2) + y;
                else
                        p = (x * z) * ((P1 + z * P2) + (z * z) * (P3 + z *
                            P4)) + y;

                        p = (x * z) * ((P1 + z * P2) + (z * z) * (P3 +
                            z * P4)) + y;
                return (x + p);
        } else {                        /* 0.164062500 < |x| < ~pi/4 */
                n = ix >> 20;
                i = (((ix >> 12) & 0xff) | 0x100) >> (0x401 - n);
                j = i - 10;

#endif /* ! codereview */
                if (hx < 0)
                        v = -y - (_TBL_sincosx[j] + x);
                else
                        v = y - (_TBL_sincosx[j] - x);

#endif /* ! codereview */
                s = v * v;
                j <<= 1;
                w = _TBL_sincos[j];
                z = _TBL_sincos[j + 1];
                z = _TBL_sincos[j+1];
                p = s * (PP1 + s * PP2);
                q = s * (QQ1 + s * QQ2);
                p = v + v * p;
                s = w * q + z * p;
                return ((hx >= 0) ? w + s : -(w + s));
                return ((hx >= 0)? w + s : -(w + s));
        }
}
```

_____*unchanged_portion_omitted_*

```
**********************************************************
    5042 Thu Jun  6 20:37:35 2019
new/usr/src/lib/libm/common/C/__sincos.c
11210 libm should be cstyle(10NBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

#endif /* ! codereview */
/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2005 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */


/* INDENT OFF */
/*
 * double __k_sincos(double x, double y, double *c);
 * kernel sincos function on [-pi/4, pi/4], pi/4 ~ 0.785398164
 * Input x is assumed to be bounded by ~pi/4 in magnitude.
 * Input y is the tail of x.
 * return sin(x) with *c = cos(x)
 *
 * Accurate Table look-up algorithm by K.C. Ng, May, 1995.
 *
 * 1. Reduce x to x>0 by sin(-x)=-sin(x),cos(-x)=cos(x).
 * 2. For 0<= x < pi/4, let i = (64*x chopped)-10. Let d = x - a[i], where
 *    a[i] is a double that is close to (i+10.5)/64 and such that
 *    sin(a[i]) and cos(a[i]) is close to a double (with error less
 *    than 2**-8 ulp). Then
 *      cos(x) = cos(a[i]+d) = cos(a[i])cos(d) - sin(a[i])*sin(d)
 *             = TBL_cos_a[i]*(1+QQ1*d^2+QQ2*d^4) -
 *                     TBL_sin_a[i]*(d+PP1*d^3+PP2*d^5)
 *             = TBL_cos_a[i] + (TBL_cos_a[i]*d^2*(QQ1+QQ2*d^2) -
 *                     TBL_sin_a[i]*(d+PP1*d^3+PP2*d^5))
 *      sin(x) = sin(a[i]+d) = sin(a[i])cos(d) + cos(a[i])*sin(d)
 *             = TBL_sin_a[i]*(1+QQ1*d^2+QQ2*d^4) +
 *                     TBL_cos_a[i]*(d+PP1*d^3+PP2*d^5)
 *             = TBL_sin_a[i] + (TBL_sin_a[i]*d^2*(QQ1+QQ2*d^2) +
 *                     TBL_cos_a[i]*(d+PP1*d^3+PP2*d^5))
 *
 *    For |y| less than 10.5/64 = 0.1640625, use
 *      sin(y) = y + y^3*(p1+y^2*(p2+y^2*(p3+y^2*p4)))
```



```
 *      cos(y) = 1 + y^2*(q1+y^2*(q2+y^2*(q3+y^2*q4)))
 *
 *    For |y| less than 0.008, use
 *      sin(y) = y + y^3*(pp1+y^2*pp2)
 *      cos(y) = 1 + y^2*(qq1+y^2*qq2)
 *
 * Accuracy:
 *      TRIG(x) returns trig(x) nearly rounded (less than 1 ulp)
 */

#include "libm.h"

static const double sc[] = {
/* ONE  = */
        1.0,
/* ONE  = */  1.0,
/* NONE = */ -1.0,

#endif /* ! codereview */
/*
 * |sin(x) - (x+pp1*x^3+pp2*x^5)| <= 2^-58.79 for |x| < 0.008
 */
/* PP1  = */-0.16666666666631655886725205237888952148062785868305556 7,
/* PP2  = */.00833331565299747232356489424846675824847537497797401 7927,

/* PP1  = */ -0.16666666666631655886725205237888952148062785868305556 7,
/* PP2  = */   .00833331565299747232356489424846675824847537497797401 7927,
/*
 * |(sin(x) - (x+p1*x^3+...+p4*x^9)|
 * |------------------------------ | <= 2^-57.63 for |x| < 0.1953125
 * |                 x             |
 */
/* P1   = */ -1.66666666666662966980521513892030158965 6e-0001,
/* P2   = */ 8.33333333239095129568399345528033637666 3e-0003,
/* P3   = */ -1.98412623799797669279155177823009840396 0e-0004,
/* P4   = */ 2.75340362485427723764998762284833035111 0e-0006,

#endif /* ! codereview */
/*
 * |cos(x) - (1+qq1*x^2+qq2*x^4)| <= 2^-55.99 for |x| <= 0.008 (0x3f80624d)
 */
/* QQ1  = */-0.49999999999754923818429119819484185427429 29,
/* QQ2  = */0.04166654290435205929454520915835764039877174 0,

/* QQ1  = */ -0.49999999999754923818429119819484185427429 29,
/* QQ2  = */  0.04166654290435205929454520915835764039877174 0,
/*
 * |cos(x) - (1+q1*x^2+...+q4*x^8)| <= 2^-55.86 for |x| <= 0.1640625 (10.5/64)
 */
/* Q1   = */ -0.5,
/* Q2   = */ 4.16666666650035070368094552086074861744 5e-0002,
/* Q3   = */ -1.38888859643697221069426629057784869600 6e-0003,
/* Q4   = */ 2.47856307885858947367951951789295349219 2e-0005,
};

/* INDENT ON */

#define ONE             sc[0]
#define NONE            sc[1]
#define PP1             sc[2]
#define PP2             sc[3]
#define P1              sc[4]
#define P2              sc[5]
#define P3              sc[6]
#define P4              sc[7]
#define QQ1             sc[8]
```

```
#define QQ2             sc[9]
#define Q1              sc[10]
#define Q2              sc[11]
#define Q3              sc[12]
#define Q4              sc[13]

extern const double _TBL_sincos[], _TBL_sincosx[];

double
__k_sincos(double x, double y, double *c)
{
__k_sincos(double x, double y, double *c) {
        double z, w, s, v, p, q;
        int i, j, n, hx, ix;

        hx = ((int *)&x)[HIWORD];
        ix = hx & ~0x80000000;

        if (ix <= 0x3fc50000) {         /* |x| < 10.5/64 = 0.164062500 */
                if (ix < 0x3e400000) {  /* |x| < 2**-27 */
                        if ((int)x == 0)
                                *c = ONE;

#endif /* ! codereview */
                        return (x + y);
                } else {
                        z = x * x;

#endif /* ! codereview */
                        if (ix < 0x3f800000) {  /* |x| < 0.008 */
                                q = z * (QQ1 + z * QQ2);
                                p = (x * z) * (PP1 + z * PP2) + y;
                        } else {
                                q = z * ((Q1 + z * Q2) + (z * z) * (Q3 + z *
                                    Q4));
                                q = z * ((Q1 + z * Q2) + (z * z) * (Q3 +
                                    z * Q4));
                                p = (x * z) * ((P1 + z * P2) + (z * z) * (P3 +
                                    z * P4)) + y;
                        }

#endif /* ! codereview */
                        *c = ONE + q;
                        return (x + p);
                }
        } else {                        /* 0.164062500 < |x| < ~pi/4 */
                n = ix >> 20;
                i = (((ix >> 12) & 0xff) | 0x100) >> (0x401 - n);
                j = i - 10;

#endif /* ! codereview */
                if (hx < 0)
                        v = -y - (_TBL_sincosx[j] + x);
                else
                        v = y - (_TBL_sincosx[j] - x);

#endif /* ! codereview */
                s = v * v;
                j <<= 1;
                w = _TBL_sincos[j];
                z = _TBL_sincos[j + 1];
                z = _TBL_sincos[j+1];
                p = s * (PP1 + s * PP2);
                q = s * (QQ1 + s * QQ2);
                p = v + v * p;
                *c = z - (w * p - z * q);
```



```
                s = w * q + z * p;
                return ((hx >= 0) ? w + s : -(w + s));
                return ((hx >= 0)? w + s : -(w + s));
        }
}
```

_____unchanged_portion_omitted_

```
**********************************************************
    5690 Thu Jun  6 20:37:36 2019
new/usr/src/lib/libm/common/C/__tan.c
11210 libm should be cstyle(10NBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */


/* INDENT OFF */
/*
 * __k_tan( double x;  double y; int k )
 * kernel tan/cotan function on [-pi/4, pi/4], pi/4 ~ 0.785398164
 * Input x is assumed to be bounded by ~pi/4 in magnitude.
 * Input y is the tail of x.
 * Input k indicate -- tan if k=0; else -1/tan
 *
 * Table look up algorithm
 *      1. by tan(-x) = -tan(x), need only to consider positive x
 *      2. if x < 5/32 = [0x3fc40000, 0] = 0.15625 , then
 *           if x < 2^-27 (hx < 0x3e400000 0), set w=x with inexact if x !=  0
 *           else
 *              z = x*x;
 *              w = x + (y+(x*z)*(t1+z*(t2+z*(t3+z*(t4+z*(t5+z*t6))))))
 *         return (k == 0)? w: 1/w;
 *      3. else
 *              ht = (hx + 0x4000)&0x7fff8000   (round x to a break point t)
 *              lt = 0
 *              i  = (hy-0x3fc40000)>>15;       (i<=64)
 *              x' = (x - t)+y                  (|x'| ~<= 2^-7)
 *         By
 *              tan(t+x')
 *                = (tan(t)+tan(x'))/(1-tan(x')tan(t))
 *         We have
 *                              sin(x')+tan(t)*(tan(t)*sin(x'))
 *                = tan(t) + ------------------------------    for k=0
 *                                  cos(x') - tan(t)*sin(x')
```



```
 *                                  cos(x') - tan(t)*sin(x')
 *                = - ----------------------------------------    for k=1
 *                        tan(t) + tan(t)*(cos(x')-1) + sin(x')
 *
 *
 *         where        tan(t) is from the table,
 *                      sin(x') = x + pp1*x^3 + pp2*x^5
 *                      cos(x') = 1 + qq1*x^2 + qq2*x^4
 */

#include "libm.h"

extern const double _TBL_tan_hi[], _TBL_tan_lo[];

#endif /* ! codereview */
static const double q[] = {
/* one  = */
        1.0,

/* one  = */  1.0,
/*
 *                          2       2       -59.56
 * |sin(x) - pp1*x*(pp2+x *(pp3+x )| <= 2        for |x|<1/64
 */
/* pp1  = */ 8.333261209690962303953121192989783594384789466 86e-0003,
/* pp2  = */ 1.200010385894389652150256805968686923814259445 26e+0002,
/* pp3  = */ -2.000017309750894511921615048777312040328979492 19e+0001,

/*
 *                    2      2        -56.19
 * |cos(x) - (1+qq1*x (qq2+x ))| <= 2        for |x|<=1/128
 */
/* qq1  = */ 4.166654863857219281975119429262122139334678649 90e-0002,
/* qq2  = */ -1.200003399213400356870806717779487371444702148 44e+0001,

/*
 * |tan(x) - PF(x)|
 * |--------------| <= 2^-58.57 for |x|<0.15625
 * |      x       |
 *
 * where (let z = x*x)
 *      PF(x) = x + (t1*x*z)(t2 + z(t3 + z))(t4 + z)(t5 + z(t6 + z))
 */
/* t1 = */ 3.719233589865168169291687050304062722716480493 55e-0003,
/* t2 = */ 6.026451203548578661184365046210587024688720703 12e+0000,
/* t3 = */ 2.426273275873981560835090931504964828491210937 50e+0000,
/* t4 = */ 2.449689839342527708510033335187472403049468994 14e+0000,
/* t5 = */ 6.070892525717679788499481219332665205001831054 69e+0000,
/* t6 = */ -2.494037569955937616583696581074036657810211181 64e+0000,
};

#define one             q[0]
#define pp1             q[1]
#define pp2             q[2]
#define pp3             q[3]
#define qq1             q[4]
#define qq2             q[5]
#define t1              q[6]
#define t2              q[7]
#define t3              q[8]
#define t4              q[9]
#define t5              q[10]
#define t6              q[11]

/* INDENT ON */
```

```
125 double
126 __k_tan(double x, double y, int k)
127 {
117 __k_tan(double x, double y, int k) {
128         double a, t, z, w = 0.0L, s, c, r, rh, xh, xl;
129         int i, j, hx, ix;

131         t = one;
132         hx = ((int *)&x)[HIWORD];
122         hx = ((int *) &x)[HIWORD];
133         ix = hx & 0x7fffffff;

135 #endif /* ! codereview */
136         if (ix < 0x3fc40000) {                  /* 0.15625 */
137                 if (ix < 0x3e400000) {          /* 2^-27 */
138                         if ((i = (int)x) == 0)  /* generate inexact */
124                         if ((i = (int) x) == 0)         /* generate inexact */
139                                 w = x;

141 #endif /* ! codereview */
142                         t = y;
143                 } else {
144                         z = x * x;
145                         t = y + (((t1 * x) * z) * (t2 + z * (t3 + z))) * ((t4 +
146                             z) * (t5 + z * (t6 + z)));
126                         t = y + (((t1 * x) * z) * (t2 + z * (t3 + z))) *
127                                 ((t4 + z) * (t5 + z * (t6 + z)));
147                         w = x + t;
148                 }

150 #endif /* ! codereview */
151                 if (k == 0)
152                         return (w);

154 #endif /* ! codereview */
155                 /*
156                  * Compute -1/(x+T) with great care
157                  * Let r = -1/(x+T), rh = r chopped to 20 bits.
158                  * Also let xh  = x+T chopped to 20 bits, xl = (x-xh)+T. Then
159                  *   -1/(x+T)   = rh + (-1/(x+T)-rh) = rh + r*(1+rh*(x+T))
160                  *              = rh + r*((1+rh*xh)+rh*xl).
161                  */
162                 rh = r = -one / w;
163                 ((int *)&rh)[LOWORD] = 0;
130                 ((int *) &rh)[LOWORD] = 0;
164                 xh = w;
165                 ((int *)&xh)[LOWORD] = 0;
132                 ((int *) &xh)[LOWORD] = 0;
166                 xl = (x - xh) + t;
167                 return (rh + r * ((one + rh * xh) + rh * xl));
168         }

170 #endif /* ! codereview */
171         j = (ix + 0x4000) & 0x7fff8000;
172         i = (j - 0x3fc40000) >> 15;
173         ((int *)&t)[HIWORD] = j;

136         ((int *) &t)[HIWORD] = j;
175         if (hx > 0)
176                 x = y - (t - x);
177         else
178                 x = -y - (t + x);

180 #endif /* ! codereview */
```



```
181         a = _TBL_tan_hi[i];
182         z = x * x;
183         s = (pp1 * x) * (pp2 + z * (pp3 + z));  /* sin(x) */
184         t = (qq1 * z) * (qq2 + z);              /* cos(x) - 1 */

186 #endif /* ! codereview */
187         if (k == 0) {
188                 w = a * s;
189                 t = _TBL_tan_lo[i] + (s + a * w) / (one - (w - t));
190                 return (hx < 0 ? -a - t : a + t);
191         } else {
192                 w = s + a * t;
193                 c = w + _TBL_tan_lo[i];
194                 t = a * s - t;

196 #endif /* ! codereview */
197                 /*
198                  * Now try to compute [(1-T)/(a+c)] accurately
199                  *
200                  * Let r = 1/(a+c), rh = (1-T)*r chopped to 20 bits.
201                  * Also let xh = a+c chopped to 20 bits, xl = (a-xh)+c. Then
202                  *      (1-T)/(a+c) = rh + ((1-T)/(a+c)-rh)
203                  *                  = rh + r*(1-T-rh*(a+c))
204                  *                  = rh + r*((1-T-rh*xh)-rh*xl)
205                  *                  = rh + r*(((1-rh*xh)-T)-rh*xl)
206                  */
207                 r = one / (a + c);
208                 rh = (one - t) * r;
209                 ((int *)&rh)[LOWORD] = 0;
141                 ((int *) &rh)[LOWORD] = 0;
210                 xh = a + c;
211                 ((int *)&xh)[LOWORD] = 0;
143                 ((int *) &xh)[LOWORD] = 0;
212                 xl = (a - xh) + c;
213                 z = rh + r * (((one - rh * xh) - t) - rh * xl);
214                 return (hx >= 0 ? -z : z);
215         }
216 }
```

*_____unchanged_portion_omitted_*

```
**********************************************************
    2120 Thu Jun  6 20:37:36 2019
new/usr/src/lib/libm/common/C/__xpg6.c
11210 libm should be cstyle(10NBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

/*LINTLIBRARY*/

/*
 * See /ws/unix200x-gate/usr/src/lib/libc/port/gen/xpg6.c for libc default.
 * __xpg6 (C99/SUSv3) is first included in Solaris 10 libc and libm
 * as well as the K2 (S1S8) libsunmath and libmopt.
 *
 * The default setting, _C99SUSv3_mode_OFF, means to retain current Solaris
 * behavior which is NOT C99/SUSv3 compliant.  This is normal.  These libraries
 * determine which standard to use based on how applications are built.  These
 * libraries at runtime determine which behavior to choose based on the value
 * of __xpg6.  By default they retain their original Solaris behavior.
 *
 * __xpg6 is used to control certain behaviors between the C99 standard, the
 * SUSv3 standard, and Solaris.  More explanation in lib/libc/inc/xpg6.h.
 * The XPG6 C compiler utility (c99) will add an object file that contains
 * an alternate definition for __xpg6.  The symbol interposition provided
 * by the linker will allow these libraries to find that symbol instead.
 *
 * Possible settings are available and documented in lib/libc/inc/xpg6.h.
 */

#include "xpg6.h"

unsigned int __xpg6 = _C99SUSv3_mode_OFF;
```

```
**********************************************************
    1203 Thu Jun  6 20:37:37 2019
new/usr/src/lib/libm/common/C/_lib_version.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

/*
 * values-{X,x}?.o should define + initialize an *actual* symbol _lib_version.
 */

#include <math.h>

#pragma weak _lib_version = __libm_lib_version

const enum version __libm_lib_version = libm_ieee;
```

```
**********************************************************
    4608 Thu Jun  6 20:37:37 2019
new/usr/src/lib/libm/common/C/acos.c
11210 libm should be cstyle(10NBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __acos = acos


/* INDENT OFF */
/*
 * acos(x)
 * Method :
 *      acos(x)  = pi/2 - asin(x)
 *      acos(-x) = pi/2 + asin(x)
 * For |x|<=0.5
 *      acos(x) = pi/2 - (x + x*x^2*R(x^2))     (see asin.c)
 * For x>0.5
 *      acos(x) = pi/2 - (pi/2 - 2asin(sqrt((1-x)/2)))
 *              = 2asin(sqrt((1-x)/2))
 *              = 2s + 2s*z*R(z)        ...z=(1-x)/2, s=sqrt(z)
 *              = 2f + (2c + 2s*z*R(z))
 *     where f=hi part of s, and c = (z-f*f)/(s+f) is the correction term
 *     for f so that f+c ~ sqrt(z).
 * For x<-0.5
 *      acos(x) = pi - 2asin(sqrt((1-|x|)/2))
 *              = pi - 0.5*(s+s*z*R(z)), where z=(1-|x|)/2,s=sqrt(z)
 *
 * Special cases:
 *      if x is NaN, return x itself;
 *      if |x|>1, return NaN with invalid signal.
 *
 * Function needed: sqrt
 */
/* INDENT ON */
```



```
#include "libm_protos.h"                /* _SVID_libm_error */
#include "libm_macros.h"
#include <math.h>

/* INDENT OFF */
static const double xxx[] = {
/* one */
        1.00000000000000000000e+00,             /* 3FF00000, 00000000 */
/* one */        1.00000000000000000000e+00,    /* 3FF00000, 00000000 */
/* pi */ 3.14159265358979311600e+00,            /* 400921FB, 54442D18 */
/* pio2_hi */ 1.57079632679489655800e+00,       /* 3FF921FB, 54442D18 */
/* pio2_lo */ 6.12323399573676603587e-17,       /* 3C91A626, 33145C07 */
/* pS0 */ 1.66666666666666657415e-01,           /* 3FC55555, 55555555 */
/* pS1 */ -3.25565818622400915405e-01,          /* BFD4D612, 03EB6F7D */
/* pS2 */ 2.01212532134862925881e-01,           /* 3FC9C155, 0E884455 */
/* pS3 */ -4.00555345006794114027e-02,          /* BFA48228, B5688F3B */
/* pS4 */ 7.91534994289814532176e-04,           /* 3F49EFE0, 7501B288 */
/* pS5 */ 3.47933107596021167570e-05,           /* 3F023DE1, 0DFDF709 */
/* qS1 */ -2.40339491173441421878e+00,          /* C0033A27, 1C8A2D4B */
/* qS2 */ 2.02094576023350569471e+00,           /* 40002AE5, 9C598AC8 */
/* qS3 */ -6.88283971605453293030e-01,          /* BFE6066C, 1B8D0159 */
/* qS4 */ 7.70381505559019352791e-02            /* 3FB3B8C5, B12E9282 */
};

#endif /* ! codereview */
#define one             xxx[0]
#define pi              xxx[1]
#define pio2_hi         xxx[2]
#define pio2_lo         xxx[3]
#define pS0             xxx[4]
#define pS1             xxx[5]
#define pS2             xxx[6]
#define pS3             xxx[7]
#define pS4             xxx[8]
#define pS5             xxx[9]
#define qS1             xxx[10]
#define qS2             xxx[11]
#define qS3             xxx[12]
#define qS4             xxx[13]
/* INDENT ON */

double
acos(double x)
{
acos(double x) {
        double z, p, q, r, w, s, c, df;
        int hx, ix;

        hx = ((int *)&x)[HIWORD];
        hx = ((int *) &x)[HIWORD];
        ix = hx & 0x7fffffff;

#endif /* ! codereview */
        if (ix >= 0x3ff00000) {         /* |x| >= 1 */
                if (((ix - 0x3ff00000) | ((int *)&x)[LOWORD]) == 0) {
                if (((ix - 0x3ff00000) | ((int *) &x)[LOWORD]) == 0) {
                        /* |x| == 1 */
                        if (hx > 0)     /* acos(1) = 0 */
                                return (0.0);
                        else            /* acos(-1) = pi */
                                return (pi + 2.0 * pio2_lo);
                } else if (isnan(x)) {
                } else if (isnan(x))
#if defined(FPADD_TRAPS_INCOMPLETE_ON_NAN)
                        return (ix >= 0x7ff80000 ? x : (x - x) / (x - x));
                /* assumes sparc-like QNaN */
```

```
#else
                        return ((x - x) / (x - x));     /* acos(|x|>1) is NaN */
                        return (x - x) / (x - x);       /* acos(|x|>1) is NaN */
#endif
                } else {
                        else
                        return (_SVID_libm_err(x, x, 1));
                }
        }

#endif /* ! codereview */
        if (ix < 0x3fe00000) {                          /* |x| < 0.5 */
                if (ix <= 0x3c600000)
                        return (pio2_hi + pio2_lo);     /* if |x| < 2**-57 */

#endif /* ! codereview */
                z = x * x;
                p = z * (pS0 + z * (pS1 + z * (pS2 + z * (pS3 + z * (pS4 + z *
                    pS5)))));
                p = z * (pS0 + z * (pS1 + z * (pS2 + z * (pS3 +
                        z * (pS4 + z * pS5)))));
                q = one + z * (qS1 + z * (qS2 + z * (qS3 + z * qS4)));
                r = p / q;
                return (pio2_hi - (x - (pio2_lo - x * r)));
        } else if (hx < 0) {
                /* x < -0.5 */
                z = (one + x) * 0.5;
                p = z * (pS0 + z * (pS1 + z * (pS2 + z * (pS3 + z * (pS4 + z *
                    pS5)))));
                p = z * (pS0 + z * (pS1 + z * (pS2 + z * (pS3 +
                        z * (pS4 + z * pS5)))));
                q = one + z * (qS1 + z * (qS2 + z * (qS3 + z * qS4)));
                s = sqrt(z);
                r = p / q;
                w = r * s - pio2_lo;
                return (pi - 2.0 * (s + w));
        } else {
                /* x > 0.5 */
                z = (one - x) * 0.5;
                s = sqrt(z);
                df = s;
                ((int *)&df)[LOWORD] = 0;
                ((int *) &df)[LOWORD] = 0;
                c = (z - df * df) / (s + df);
                p = z * (pS0 + z * (pS1 + z * (pS2 + z * (pS3 + z * (pS4 + z *
                    pS5)))));
                p = z * (pS0 + z * (pS1 + z * (pS2 + z * (pS3 +
                        z * (pS4 + z * pS5)))));
                q = one + z * (qS1 + z * (qS2 + z * (qS3 + z * qS4)));
                r = p / q;
                w = r * s + c;
                return (2.0 * (df + w));
        }
}
```

*_____unchanged_portion_omitted_*

```
**********************************************************
    2540 Thu Jun  6 20:37:38 2019
new/usr/src/lib/libm/common/C/acosh.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __acosh = acosh


/* INDENT OFF */
/*
 * acosh(x)
 * Method :
 *      Based on
 *                acosh(x) = log [ x + sqrt(x*x-1) ]
 *      we have
 *                acosh(x) := log(x)+ln2, if x is large; else
 *                acosh(x) := log(2x-1/(sqrt(x*x-1)+x)) if x > 2; else
 *                acosh(x) := log1p(t+sqrt(2.0*t+t*t)); where t = x-1.
 *
 * Special cases:
 *      acosh(x) is NaN with signal if x < 1.
 *      acosh(NaN) is NaN without signal.
 */
/* INDENT ON */

#include "libm_protos.h"                /* _SVID_libm_error */
#include "libm_macros.h"
#include <math.h>

static const double one = 1.0,
static const double
        one = 1.0,
        ln2 = 6.93147180559945286227e-01;       /* 3FE62E42, FEFA39EF */

double
```



```
acosh(double x)
{
acosh(double x) {
        double t;
        int hx;

        hx = ((int *)&x)[HIWORD];

        hx = ((int *) &x)[HIWORD];
        if (hx < 0x3ff00000) {          /* x < 1 */
                if (isnan(x))
#if defined(FPADD_TRAPS_INCOMPLETE_ON_NAN)
                        return (hx >= 0xfff80000 ? x : (x - x) / (x - x));

#endif /* ! codereview */
                /* assumes sparc-like QNaN */
#else
                        return ((x - x) / (x - x));
                        return (x - x) / (x - x);
#endif
                else
                        return (_SVID_libm_err(x, x, 29));
        } else if (hx >= 0x41b00000) {
                /* x > 2**28 */
                if (hx >= 0x7ff00000) { /* x is inf of NaN */
#if defined(FPADD_TRAPS_INCOMPLETE_ON_NAN)
                        return (hx >= 0x7ff80000 ? x : x + x);
                        /* assumes sparc-like QNaN */
#else
                        return (x + x);
#endif
                } else { /* acosh(huge)=log(2x) */
                } else  /* acosh(huge)=log(2x) */
                        return (log(x) + ln2);
                }
        } else if (((hx - 0x3ff00000) | ((int *)&x)[LOWORD]) == 0) {
        } else if (((hx - 0x3ff00000) | ((int *) &x)[LOWORD]) == 0) {
                return (0.0);           /* acosh(1) = 0 */
        } else if (hx > 0x40000000) {
                /* 2**28 > x > 2 */
                t = x * x;
                return (log(2.0 * x - one / (x + sqrt(t - one))));
        } else {
                /* 1 < x < 2 */
                t = x - one;
                return (log1p(t + sqrt(2.0 * t + t * t)));
        }
}
______unchanged_portion_omitted_
```

```
**********************************************************
    4793 Thu Jun  6 20:37:38 2019
new/usr/src/lib/libm/common/C/asin.c
11210 libm should be cstyle(10NBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __asin = asin


/* INDENT OFF */
/*
 * asin(x)
 * Method :
 *      Since  asin(x) = x + x^3/6 + x^5*3/40 + x^7*15/336 + ...
 *      we approximate asin(x) on [0,0.5] by
 *              asin(x) = x + x*x^2*R(x^2)
 *      where
 *              R(x^2) is a rational approximation of (asin(x)-x)/x^3
 *      and its remez error is bounded by
 *              |(asin(x)-x)/x^3 - R(x^2)| < 2^(-58.75)
 *
 *      For x in [0.5,1]
 *              asin(x) = pi/2-2*asin(sqrt((1-x)/2))
 *      Let y = (1-x), z = y/2, s := sqrt(z), and pio2_hi+pio2_lo=pi/2;
 *      then for x>0.98
 *              asin(x) = pi/2 - 2*(s+s*z*R(z))
 *                      = pio2_hi - (2*(s+s*z*R(z)) - pio2_lo)
 *      For x<=0.98, let pio4_hi = pio2_hi/2, then
 *              f = hi part of s;
 *              c = sqrt(z) - f = (z-f*f)/(s+f)         ...f+c=sqrt(z)
 *      and
 *              asin(x) = pi/2 - 2*(s+s*z*R(z))
 *                      = pio4_hi+(pio4-2s)-(2s*z*R(z)-pio2_lo)
 *                      = pio4_hi+(pio4-2f)-(2s*z*R(z)-(pio2_lo+2c))
 *
 * Special cases:
```



```
 *      if x is NaN, return x itself;
 *      if |x|>1, return NaN with invalid signal.
 *
 */
/* INDENT ON */

#include "libm_protos.h"                /* _SVID_libm_error */
#include "libm_macros.h"
#include <math.h>

/* INDENT OFF */
static const double xxx[] = {
/* one */
        1.00000000000000000000e+00,             /* 3FF00000, 00000000 */
/* one */        1.00000000000000000000e+00,    /* 3FF00000, 00000000 */
/* huge */ 1.000e+300,
/* pio2_hi */ 1.57079632679489655800e+00,       /* 3FF921FB, 54442D18 */
/* pio2_lo */ 6.12323399573676603587e-17,       /* 3C91A626, 33145C07 */
/* pio4_hi */ 7.85398163397448278999e-01,       /* 3FE921FB, 54442D18 */

/*
 * coefficient for R(x^2)
 * pS0
 */
        1.66666666666666657415e-01,             /* 3FC55555, 55555555 */
/* coefficient for R(x^2) */
/* pS0 */        1.66666666666666657415e-01,    /* 3FC55555, 55555555 */
/* pS1 */ -3.25565818622400915405e-01,          /* BFD4D612, 03EB6F7D */
/* pS2 */ 2.01212532134862925881e-01,           /* 3FC9C155, 0E884455 */
/* pS3 */ -4.00555345006794114027e-02,          /* BFA48228, B5688F3B */
/* pS4 */ 7.91534994289814532176e-04,           /* 3F49EFE0, 7501B288 */
/* pS5 */ 3.47933107596021167570e-05,           /* 3F023DE1, 0DFDF709 */
/* qS1 */ -2.40339491173441421878e+00,          /* C0033A27, 1C8A2D4B */
/* qS2 */ 2.02094576023350569471e+00,           /* 40002AE5, 9C598AC8 */
/* qS3 */ -6.88283971605453293030e-01,          /* BFE6066C, 1B8D0159 */
/* qS4 */ 7.70381505559019352791e-02            /* 3FB3B8C5, B12E9282 */
};

#endif /* ! codereview */
#define one             xxx[0]
#define huge            xxx[1]
#define pio2_hi         xxx[2]
#define pio2_lo         xxx[3]
#define pio4_hi         xxx[4]
#define pS0             xxx[5]
#define pS1             xxx[6]
#define pS2             xxx[7]
#define pS3             xxx[8]
#define pS4             xxx[9]
#define pS5             xxx[10]
#define qS1             xxx[11]
#define qS2             xxx[12]
#define qS3             xxx[13]
#define qS4             xxx[14]
/* INDENT ON */

double
asin(double x)
{
asin(double x) {
        double t, w, p, q, c, r, s;
        int hx, ix, i;

        hx = ((int *)&x)[HIWORD];
        hx = ((int *) &x)[HIWORD];
        ix = hx & 0x7fffffff;
```

```
#endif /* ! codereview */
        if (ix >= 0x3ff00000) {         /* |x| >= 1 */
                if (((ix - 0x3ff00000) | ((int *)&x)[LOWORD]) == 0)
                if (((ix - 0x3ff00000) | ((int *) &x)[LOWORD]) == 0)
                        /* asin(1)=+-pi/2 with inexact */
                        return (x * pio2_hi + x * pio2_lo);
                else if (isnan(x))
#if defined(FPADD_TRAPS_INCOMPLETE_ON_NAN)
                        return (ix >= 0x7ff80000 ? x : (x - x) / (x - x));

#endif /* ! codereview */
                /* assumes sparc-like QNaN */
#else
                        return ((x - x) / (x - x));     /* asin(|x|>1) is NaN */
                        return (x - x) / (x - x);       /* asin(|x|>1) is NaN */
#endif
                else
                        return (_SVID_libm_err(x, x, 2));
        } else if (ix < 0x3fe00000) {   /* |x| < 0.5 */
                if (ix < 0x3e400000) {  /* if |x| < 2**-27 */
                        if ((i = (int)x) == 0)
                        if ((i = (int) x) == 0)
                                /* return x with inexact if x != 0 */
                                return (x);
                }

#endif /* ! codereview */
                t = x * x;
                p = t * (pS0 + t * (pS1 + t * (pS2 + t * (pS3 + t * (pS4 + t *
                    pS5)))));
                p = t * (pS0 + t * (pS1 + t * (pS2 + t * (pS3 +
                        t * (pS4 + t * pS5)))));
                q = one + t * (qS1 + t * (qS2 + t * (qS3 + t * qS4)));
                w = p / q;
                return (x + x * w);
        }

#endif /* ! codereview */
        /* 1 > |x| >= 0.5 */
        w = one - fabs(x);
        t = w * 0.5;
        p = t * (pS0 + t * (pS1 + t * (pS2 + t * (pS3 + t * (pS4 + t * pS5)))));
        q = one + t * (qS1 + t * (qS2 + t * (qS3 + t * qS4)));
        s = sqrt(t);

#endif /* ! codereview */
        if (ix >= 0x3FEF3333) {         /* if |x| > 0.975 */
                w = p / q;
                t = pio2_hi - (2.0 * (s + s * w) - pio2_lo);
        } else {
                w = s;
                ((int *)&w)[LOWORD] = 0;
                ((int *) &w)[LOWORD] = 0;
                c = (t - w * w) / (s + w);
                r = p / q;
                p = 2.0 * s * r - (pio2_lo - 2.0 * c);
                q = pio4_hi - 2.0 * w;
                t = pio4_hi - (p - q);
        }

#endif /* ! codereview */
        return (hx > 0 ? t : -t);
}
```

```
**********************************************************
    2373 Thu Jun  6 20:37:39 2019
new/usr/src/lib/libm/common/C/asinh.c
11210 libm should be cstyle(10NBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __asinh = asinh


/* INDENT OFF */
/*
 * asinh(x)
 * Method :
 *      Based on
 *                asinh(x) = sign(x) * log [ |x| + sqrt(x*x+1) ]
 *      we have
 *      asinh(x) := x  if  1+x*x == 1,
 *               := sign(x)*(log(x)+ln2)) for large |x|, else
 *               := sign(x)*log(2|x|+1/(|x|+sqrt(x*x+1))) if|x| > 2, else
 *               := sign(x)*log1p(|x|+x^2/(1+sqrt(1+x^2)))
 */
/* INDENT ON */

#include "libm_macros.h"
#include <math.h>

static const double xxx[] = {
/* one */
        1.00000000000000000000e+00,     /* 3FF00000, 00000000 */
/* one */       1.00000000000000000000e+00,     /* 3FF00000, 00000000 */
/* ln2 */ 6.93147180559945286227e-01,   /* 3FE62E42, FEFA39EF */
/* huge */ 1.00000000000000000000e+300
};

#endif /* ! codereview */
#define one             xxx[0]
```



```
#define ln2             xxx[1]
#define huge            xxx[2]

double
asinh(double x)
{
asinh(double x) {
        double t, w;
        int hx, ix;

        hx = ((int *)&x)[HIWORD];
        hx = ((int *) &x)[HIWORD];
        ix = hx & 0x7fffffff;

#endif /* ! codereview */
        if (ix >= 0x7ff00000)
#if defined(FPADD_TRAPS_INCOMPLETE_ON_NAN)
                return (ix >= 0x7ff80000 ? x : x + x);

#endif /* ! codereview */
        /* assumes sparc-like QNaN */
#else
                return (x + x);         /* x is inf or NaN */
#endif

#endif /* ! codereview */
        if (ix < 0x3e300000) {          /* |x|<2**-28 */
                if (huge + x > one)
                        return (x);     /* return x inexact except 0 */
        }

#endif /* ! codereview */
        if (ix > 0x41b00000) {          /* |x| > 2**28 */
                w = log(fabs(x)) + ln2;
        } else if (ix > 0x40000000) {
                /* 2**28 > |x| > 2.0 */
                t = fabs(x);
                w = log(2.0 * t + one / (sqrt(x * x + one) + t));
        } else {
                /* 2.0 > |x| > 2**-28 */
                t = x * x;
                w = log1p(fabs(x) + t / (one + sqrt(one + t)));
        }

#endif /* ! codereview */
        return (hx > 0 ? w : -w);
}
```

```
**********************************************************
    5992 Thu Jun  6 20:37:39 2019
new/usr/src/lib/libm/common/C/atan.c
11210 libm should be cstyle(10NBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __atan = atan


/* INDENT OFF */
/*
 * atan(x)
 * Accurate Table look-up algorithm with polynomial approximation in
 * partially product form.
 *
 * -- K.C. Ng, October 17, 2004
 *
 * Algorithm
 *
 * (1). Purge off Inf and NaN and 0
 * (2). Reduce x to positive by atan(x) = -atan(-x).
 * (3). For x <= 1/8 and let z = x*x, return
 *      (2.1) if x < 2^(-prec/2), atan(x) = x  with inexact flag raised
 *      (2.2) if x < 2^(-prec/4-1), atan(x) = x+(x/3)(x*x)
 *      (2.3) if x < 2^(-prec/6-2), atan(x) = x+(z-5/3)(z*x/5)
 *      (2.4) Otherwise
 *              atan(x) = poly1(x) = x + A * B,
 *      where
 *              A = (p1*x*z) * (p2+z(p3+z))
 *              B = (p4+z)+z*z) * (p5+z(p6+z))
 *      Note: (i) domain of poly1 is [0, 1/8], (ii) remez relative
 *      approximation error of poly1 is bounded by
 *              |(atan(x)-poly1(x))/x| <= 2^-57.61
 * (4). For x >= 8 then
 *      (3.1) if x >= 2^prec,     atan(x) = atan(inf) - pio2lo
 *      (3.2) if x >= 2^(prec/3), atan(x) = atan(inf) - 1/x
```



```
 *      (3.3) if x <= 65,         atan(x) = atan(inf) - poly1(1/x)
 *      (3.4) otherwise           atan(x) = atan(inf) - poly2(1/x)
 *      where
 *              poly2(r) = (q1*r) * (q2+z(q3+z)) * (q4+z),
 *      its domain is [0, 0.0154]; and its remez absolute
 *      approximation error is bounded by
 *              |atan(x)-poly2(x)|<= 2^-59.45
 *
 * (5). Now x is in (0.125, 8).
 *      Recall identity
 *              atan(x) = atan(y) + atan((x-y)/(1+x*y)).
 *      Let j = (ix - 0x3fc00000) >> 16, 0 <= j < 96, where ix is the high
 *      part of x in IEEE double format. Then
 *              atan(x) = atan(y[j]) + poly2((x-y[j])/(1+x*y[j]))
 *      where y[j] are carefully chosen so that it matches x to around 4.5
 *      bits and at the same time atan(y[j]) is very close to an IEEE double
 *      floating point number. Calculation indicates that
 *              max|(x-y[j])/(1+x*y[j])| < 0.0154
 *              j,x
 *
 * Accuracy: Maximum error observed is bounded by 0.6 ulp after testing
 * more than 10 million random arguments
 */
/* INDENT ON */

#include "libm.h"
#include "libm_protos.h"

extern const double _TBL_atan[];

#endif /* ! codereview */
static const double g[] = {
/* one  = */
        1.0,
/* p1   = */8.021766242547659353512301549926633015275001525 88e-0002,
/* p2   = */1.272234217005594025806658464716747403144836425 78e+0000,
/* p3   = */-1.206069018005036408425212357542477548122406005 86e+0000,
/* p4   = */-2.360889679223255654960667015984654426574707031 25e+0000,
/* p5   = */1.383457995013891661528759868815541267395019531 25e+0000,
/* p6   = */1.067423680789534534696372247708495706319808959 96e+0000,
/* one  = */  1.0,
/* p1   = */  8.021766242547659353512301549926633015275001525 88e-0002,
/* p2   = */  1.272234217005594025806658464716747403144836425 78e+0000,
/* p3   = */ -1.206069018005036408425212357542477548122406005 86e+0000,
/* p4   = */ -2.360889679223255654960667015984654426574707031 25e+0000,
/* p5   = */  1.383457995013891661528759868815541267395019531 25e+0000,
/* p6   = */  1.067423680789534534696372247708495706319808959 96e+0000,
/* q1   = */ -1.427966263339117969355385184826445765793323516 85e-0001,
/* q2   = */ 3.514271104478732270598104771598634970780596291 2e+0000,
/* q3   = */ 5.921291127081642624574442379525862634181976318 36e-0001,
/* q4   = */ -1.992722347856831444090630611754022538661956787 11e+0000,
/* pio2hi */ 1.570796326794896558e+00,
/* pio2lo */ 6.123233995736765886e-17,
/* t1   = */ -0.33333333333333333333333333333333,
/* t2   = */ 0.2,
/* t3   = */ -1.66666666666666666666666666666666,
};

#define one             g[0]
#define p1              g[1]
#define p2              g[2]
#define p3              g[3]
#define p4              g[4]
#define p5              g[5]
#define p6              g[6]
#define q1              g[7]
```

```
119 #define q2              g[8]
120 #define q3              g[9]
121 #define q4              g[10]
122 #define pio2hi          g[11]
123 #define pio2lo          g[12]
124 #define t1              g[13]
125 #define t2              g[14]
126 #define t3              g[15]


128 double
129 atan(double x)
130 {
118 atan(double x) {
131         double y, z, r, p, s;
132         int ix, lx, hx, j;

134         hx = ((int *)&x)[HIWORD];
135         lx = ((int *)&x)[LOWORD];
122         hx = ((int *) &x)[HIWORD];
123         lx = ((int *) &x)[LOWORD];
136         ix = hx & ~0x80000000;
137         j = ix >> 20;

139         /* for |x| < 1/8 */
140         if (j < 0x3fc) {
141                 if (j < 0x3f5) {                /* when |x| < 2**(-prec/6-2) */
142                         if (j < 0x3e3)          /* if |x| < 2**(-prec/2-2) */
143                                 return ((int)x == 0 ? x : one);

130                         if (j < 0x3e3) {        /* if |x| < 2**(-prec/2-2) */
131                                 return ((int) x == 0 ? x : one);
132                         }
145                         if (j < 0x3f1) {        /* if |x| < 2**(-prec/4-1) */
146                                 return (x + (x * t1) * (x * x));
147                         } else {                /* if |x| < 2**(-prec/6-2) */
148                                 z = x * x;
149                                 s = t2 * x;
150                                 return (x + (t3 + z) * (s * z));
151                         }
152                 }

154                 z = x * x;
155                 s = p1 * x;
141                 z = x * x; s = p1 * x;
156                 return (x + ((s * z) * (p2 + z * (p3 + z))) *
157                     (((p4 + z) + z * z) * (p5 + z * (p6 + z))));
158         }

160         /* for |x| >= 8.0 */
161         if (j >= 0x402) {
162                 if (j < 0x436) {
163                         r = one / x;

165 #endif /* ! codereview */
166                         if (hx >= 0) {
167                                 y = pio2hi;
168                                 p = pio2lo;
150                                 y =  pio2hi; p =  pio2lo;
169                         } else {
170                                 y = -pio2hi;
171                                 p = -pio2lo;
152                                 y = -pio2hi; p = -pio2lo;
172                         }

174 #endif /* ! codereview */
```



```
175                         if (ix < 0x40504000) {  /* x <  65 */
176                                 z = r * r;
177                                 s = p1 * r;
178                                 return (y + ((p - r) - ((s * z) * (p2 + z *
179                                     (p3 + z))) * (((p4 + z) + z * z) *
154                                 return (y + ((p - r) - ((s * z) *
155                                         (p2 + z * (p3 + z))) *
156                                         (((p4 + z) + z * z) *
180                                     (p5 + z * (p6 + z)))));
181                         } else if (j < 0x412) {
182                                 z = r * r;
183                                 return (y + (p - ((q1 * r) * (q4 + z)) * (q2 +
184                                     z * (q3 + z))));
185                         } else {
160                                 return (y + (p - ((q1 * r) * (q4 + z)) *
161                                         (q2 + z * (q3 + z))));
162                         } else
186                                 return (y + (p - r));
187                         }
188 #endif /* ! codereview */
189                 } else {
190                         if (j >= 0x7ff) /* x is inf or NaN */
191                                 if (((ix - 0x7ff00000) | lx) != 0)
192 #if defined(FPADD_TRAPS_INCOMPLETE_ON_NAN)
193                                         return (ix >= 0x7ff80000 ? x : x - x);

195 #endif /* ! codereview */
196                         /* assumes sparc-like QNaN */
197 #else
198                                         return (x - x);
199 #endif
200                         y = -pio2lo;
201                         return (hx >= 0 ? pio2hi - y : y - pio2hi);
202                 }
203         } else {                        /* now x is between 1/8 and 8 */
204                 double *w, w0, w1, s, z;

206                 w = (double *)_TBL_atan + (((ix - 0x3fc00000) >> 16) << 1);
207                 w0 = (hx >= 0) ? w[0] : -w[0];
164                 w = (double *) _TBL_atan + (((ix - 0x3fc00000) >> 16) << 1);
165                 w0 = (hx >= 0)? w[0] : -w[0];
208                 s = (x - w0) / (one + x * w0);
209                 w1 = (hx >= 0) ? w[1] : -w[1];
167                 w1 = (hx >= 0)? w[1] : -w[1];
210                 z = s * s;
211                 return (((q1 * s) * (q4 + z)) * (q2 + z * (q3 + z)) + w1);
212         }
213 }
```

_____*unchanged_portion_omitted_*

```
**********************************************************
   18657 Thu Jun  6 20:37:40 2019
new/usr/src/lib/libm/common/C/atan2.c
11210 libm should be cstyle(10NBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

#endif /* ! codereview */
/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __atan2 = atan2

#include "libm.h"

/*
 * Let t(0) = 1 and for i = 1, ..., 160, let t(i) be the slope of
 * the line bisecting the conical hull of the set of points (x,y)
 * where x and y are positive normal floating point numbers and
 * the high order words hx and hy of their binary representations
 * satisfy |hx - hy - i * 0x8000| <= 0x4000.  Then:
 *
 * TBL[4*i+2] is t(i) rounded to 21 significant bits (i.e., the
 *   low order word is zero), and
 *
 * TBL[4*i] + TBL[4*i+1] form a doubled-double approximation to
 *   atan(TBL[4*i+2]).
 *
 * Finally, TBL[4*161] = TBL[4*161+1] = TBL[4*161+2] = 0.
 *
 * Now for any (x,y) with 0 < y <= x and any 0 < t <= 1,  we have
 * atan(y/x) = atan(t) + atan((y-t*x)/(x+t*y)).  By choosing t =
 * TBL[4*i+2], where i is the multiple of 0x8000 nearest hx - hy,
 * if this multiple is less than 161, and i = 161 otherwise, we
 * find that |(y-t*x)/(x+t*y)| <~ 2^-5.
 */
static const double TBL[] = {
        7.8539816339744827900e-01, +3.0616169978683830179e-17,
        1.0000000000000000000e+00, +0,
        7.7198905126506112140e-01, +2.6989956960083153652e-16,
```



```
        9.7353506088256835938e-01, +0,
        7.6068143954461309164e-01, -3.5178810518941914972e-16,
        9.5174932479858398438e-01, +0,
        7.4953661876353638860e-01, -3.2548100004524337476e-16,
        9.3073129653930664062e-01, +0,
        7.3854614984728339522e-01, -2.0775571023910406668e-16,
        9.1042709350585937500e-01, +0,
        7.2770146962041337702e-01, +3.8883249403168348802e-16,
        8.9078664779663085938e-01, +0,
        7.1699492488093774512e-01, -4.0468841511547224071e-16,
        8.7176513671875000000e-01, +0,
        7.0641813488653149022e-01, +5.6902424353981484031e-17,
        8.5331964492797851562e-01, +0,
        6.9596351101035658360e-01, +2.8245513321075021303e-16,
        8.3541154861450195312e-01, +0,
        6.8562363680534943455e-01, -4.2316970721658854064e-16,
        8.1800508499145507812e-01, +0,
        6.7539055666438230219e-01, +4.3535917281300047233e-16,
        8.0106592178344726562e-01, +0,
        6.6525763346931832132e-01, +1.1830431602404727977e-17,
        7.8456401824951171875e-01, +0,
        6.5521767574310185722e-01, -1.7435923100651044208e-16,
        7.6847028732299804688e-01, +0,
        6.4526390999481897381e-01, -1.4741927403093983947e-16,
        7.5275802612304687500e-01, +0,
        6.3538979894204850041e-01, +1.5734535069995660853e-16,
        7.3740243911743164062e-01, +0,
        6.2558914346942717799e-01, -2.8175588856316910960e-16,
        7.2238063812255859375e-01, +0,
        6.1585586476157949676e-01, -4.3056167357725226449e-16,
        7.0767116546630859375e-01, +0,
        6.0618408027576098362e-01, +1.5018013918429320289e-16,
        6.9325399398803710938e-01, +0,
        5.9656817827486730010e-01, +5.5271942033557644157e-17,
        6.7911052703857421875e-01, +0,
        5.8700289083426504533e-01, -8.2411369282676383293e-17,
        6.6522359848022460938e-01, +0,
        5.7748303053627658699e-01, +4.9400383775709159558e-17,
        6.5157699584960937500e-01, +0,
        5.6800353968303252117e-01, +2.9924431103311109543e-16,
        6.3815546035766601562e-01, +0,
        5.5855953863493823519e-01, -2.0306003403868777403e-16,
        6.2494468688964843750e-01, +0,
        5.4914706708329674711e-01, +2.8255378613779667461e-17,
        6.1193227767944335938e-01, +0,
        5.3976176660618069292e-01, +1.6370248781078747995e-16,
        5.9910583496093750000e-01, +0,
        5.3039888601412332747e-01, -7.6196097360093680134e-17,
        5.8645296096801757812e-01, +0,
        5.2105543924318808990e-01, -2.2400815668154739561e-16,
        5.7396411895751953125e-01, +0,
        5.1172778873967050828e-01, -3.6888136019899681185e-16,
        5.6162929534912109375e-01, +0,
        5.0241199666452196482e-01, -2.5412891474397011281e-16,
        5.4943847656250000000e-01, +0,
        4.9310493954293743712e-01, +4.4132186128251152229e-16,
        5.3738307952880859375e-01, +0,
        4.8380436844750995817e-01, -2.7844387907776656488e-16,
        5.2545595169067382812e-01, +0,
        4.7450670361463753721e-01, -2.0494355197368286028e-16,
        5.1364850997924804688e-01, +0,
        4.6367660027976320691e-01, +3.1709878607954760668e-16,
        5.0003623962402343750e-01, +0,
        4.5304753104003925301e-01, +3.3593436122420574865e-16,
        4.8681926727294921875e-01, +0,
        4.4423658037407065535e-01, +2.1987183192008082015e-17,
```

```
        4.7596645355224609375e-01, +0,
        4.3567016972500294258e-01, +3.0118422805369552650e-16,
        4.6550178527832031250e-01, +0,
        4.2733152672544871820e-01, -3.2667693224866479909e-16,
        4.5539522171020507812e-01, +0,
        4.1920540176693954493e-01, -2.2454273841113897647e-16,
        4.4561982154846191406e-01, +0,
        4.1127722812701872357e-01, -3.1620568973494653391e-16,
        4.3615055084228515625e-01, +0,
        4.0353384063084263289e-01, -3.5932009901481421723e-16,
        4.2696499824523925781e-01, +0,
        3.9596319345246833166e-01, -4.0281533417458698585e-16,
        4.1804289817810058594e-01, +0,
        3.8855405220339722661e-01, +1.6132231486045176674e-16,
        4.0936565399169921875e-01, +0,
        3.8129566313738116889e-01, +1.7684657060650804570e-16,
        4.0091586112976074219e-01, +0,
        3.7417884791401867517e-01, +2.6897604227426977619e-16,
        3.9267849922180175781e-01, +0,
        3.6719421967585041955e-01, -4.5886151448673745001e-17,
        3.8463878631591796875e-01, +0,
        3.6033388248727771241e-01, +1.5804115573136074946e-16,
        3.7678408622741699219e-01, +0,
        3.5358982224579182940e-01, +1.2624619863035782939e-16,
        3.6910200119018554688e-01, +0,
        3.4695498404186952968e-01, +9.3221684607372865177e-17,
        3.6158156394958496094e-01, +0,
        3.4042268308109679964e-01, +2.7697913559445449137e-16,
        3.5421252250671386719e-01, +0,
        3.3398684598563566084e-01, +3.6085337449716011085e-16,
        3.4698557853698730469e-01, +0,
        3.2764182824591436827e-01, +2.0581506352606456186e-16,
        3.3989214897155761719e-01, +0,
        3.2138200938788497041e-01, -1.9015787485430693661e-16,
        3.3292388916015625000e-01, +0,
        3.1520245348069497737e-01, +2.6961839659264087022e-16,
        3.2607340812683105469e-01, +0,
        3.0909871873117023000e-01, -1.5641891686756272625e-16,
        3.1933403015136718750e-01, +0,
        3.0306644308947827682e-01, +2.8801634211591956223e-16,
        3.1269931793212890625e-01, +0,
        2.9710135482774191473e-01, -4.3148994478973365819e-16,
        3.0616307258605957031e-01, +0,
        2.9120015759141004708e-01, -6.8539854790808585159e-17,
        2.9972028732299804688e-01, +0,
        2.8535879880370362827e-01, -1.2231638445300492682e-16,
        2.9336524009704589844e-01, +0,
        2.7957422506893880865e-01, -4.6707752931043135528e-17,
        2.8709340095520019531e-01, +0,
        2.7384352102802367313e-01, -4.1215636366229625876e-16,
        2.8090047836303710938e-01, +0,
        2.6816369484161040049e-01, -2.3700583122400495333e-16,
        2.7478218078613281250e-01, +0,
        2.6253212627627764419e-01, +2.3123213692190889610e-16,
        2.6873469352722167969e-01, +0,
        2.5694635355759309903e-01, -4.0638513814701264145e-16,
        2.6275444030761718750e-01, +0,
        2.5140385572454615470e-01, -3.4795333793554943723e-16,
        2.5683784484863281250e-01, +0,
        2.4500357070096612233e-01, +6.6542334848010259289e-17,
        2.5002646446228027344e-01, +0,
        2.3877766609573036760e-01, -2.7756633678549343650e-16,
        2.4342155456542968750e-01, +0,
        2.3365669377188336142e-01, +3.2700803838522067998e-16,
        2.3800384998321533203e-01, +0,
        2.2870810463931334766e-01, -4.4279127662219799521e-16,
```



```
        2.3278105258941650391e-01, +0,
        2.2391820542294382790e-01, +3.7558889374284208052e-16,
        2.2773718833923339844e-01, +0,
        2.1927501815429550902e-01, -1.4829838176513811186e-16,
        2.2285830974578857422e-01, +0,
        2.1476740847367459253e-01, -2.0535381496063397578e-17,
        2.1813154220581054688e-01, +0,
        2.1038568111737454558e-01, -4.2826767738736168650e-16,
        2.1354568004608154297e-01, +0,
        2.0612057974373865221e-01, +4.2108051749502232359e-16,
        2.0909011363983154297e-01, +0,
        2.0196410359405447821e-01, +3.5157118083511092869e-16,
        2.0475566387176513672e-01, +0,
        1.9790861144712756925e-01, +3.7894950972257700994e-16,
        2.0053362846374511719e-01, +0,
        1.9394752160084305359e-01, +2.8270367403478935534e-16,
        1.9641649723052978516e-01, +0,
        1.9007440763641536563e-01, -2.0842758095683676397e-16,
        1.9239699840545654297e-01, +0,
        1.8628369629742813629e-01, +3.4710917040399448932e-16,
        1.8846881389617919922e-01, +0,
        1.8256998712939509488e-01, +1.1053834120570125251e-16,
        1.8462586402893066406e-01, +0,
        1.7892875067284830237e-01, +3.0486232913366680305e-16,
        1.8086302280426025391e-01, +0,
        1.7535529778449010507e-01, -2.3810135019970148624e-16,
        1.7717504501342773438e-01, +0,
        1.7184559192514736736e-01, +5.1432582846210893916e-17,
        1.7355740070343017578e-01, +0,
        1.6839590847744290159e-01, +3.1605623296041433586e-18,
        1.7000591754913330078e-01, +0,
        1.6500283902547518977e-01, +1.5405422268770998251e-16,
        1.6651678085327148438e-01, +0,
        1.6166306303174859949e-01, +4.0042241517254928672e-16,
        1.6308629512786865234e-01, +0,
        1.5837358268281231943e-01, -2.2786616251622967291e-16,
        1.5971112251281738281e-01, +0,
        1.5513160990288810126e-01, -3.7547723514797166336e-16,
        1.5638816356658935547e-01, +0,
        1.5193468535499299321e-01, +4.3497510505554267446e-16,
        1.5311467647552490234e-01, +0,
        1.4878033155427861089e-01, -2.3102860235324261895e-16,
        1.4988791942596435547e-01, +0,
        1.4566628729590647140e-01, +9.9227592950040279415e-17,
        1.4670538902282714844e-01, +0,
        1.4259050967286590605e-01, -3.3869909683813096906e-18,
        1.4356482028961181641e-01, +0,
        1.3955105903633846509e-01, +1.5500435650773331566e-17,
        1.4046406745910644531e-01, +0,
        1.3654610022831903393e-01, +3.3965918616682805753e-16,
        1.3740110397338867188e-01, +0,
        1.3357402082462854764e-01, +2.7572431581527535421e-16,
        1.3437414169311523438e-01, +0,
        1.3063319828908959153e-01, -3.4667213797076707331e-16,
        1.3138139247894287109e-01, +0,
        1.2772200049776749609e-01, +3.1089261947725651968e-16,
        1.2842106819152832031e-01, +0,
        1.2436931430778752627e-01, -4.0654251891464630059e-16,
        1.2501454353332519531e-01, +0,
        1.2111683701666819957e-01, -3.9381654342464836012e-16,
        1.2171256542205810547e-01, +0,
        1.1844801833536511282e-01, -3.6673155595150283444e-16,
        1.1900508403778076172e-01, +0,
        1.1587365536613614125e-01, -1.5026628801318421951e-16,
        1.1639505624771118164e-01, +0,
        1.1338607085741525538e-01, +1.2886806274050538880e-16,
```

```
        1.1387449502944946289e-01, +0,
        1.1097844020819369604e-01, +2.3848343623577768044e-16,
        1.1143630743026733398e-01, +0,
        1.0864456107308662069e-01, +4.2065430313285469408e-16,
        1.0907405614852905273e-01, +0,
        1.0637891628473727934e-01, -4.6883543790348472687e-18,
        1.0678201913833618164e-01, +0,
        1.0417650062205296990e-01, +1.4774925414624453292e-16,
        1.0455501079559326172e-01, +0,
        1.0203276464730581807e-01, -1.5677032794816452332e-16,
        1.0238832235336303711e-01, +0,
        9.9943617083734892503e-02, +3.4511310907979792828e-16,
        1.0027772188186645508e-01, +0,
        9.7905249824711049200e-02, +3.4489485563461708496e-16,
        9.8219275474548339844e-02, +0,
        9.5914316649349906641e-02, -1.3214510886789011569e-17,
        9.6209526062011718750e-02, +0,
        9.3967698614664918466e-02, +1.1048427091217964090e-16,
        9.4245254993438720703e-02, +0,
        9.2062564267554769515e-02, -3.7297463814697759309e-16,
        9.2323541641235351562e-02, +0,
        9.0196252506350660383e-02, -3.5280143043576718079e-16,
        9.0441644191741943359e-02, +0,
        8.8366391663268650802e-02, -6.1140673227541621183e-17,
        8.8597118854522705078e-02, +0,
        8.6570782100201526532e-02, -2.0998844594957629702e-16,
        8.6787700653076171875e-02, +0,
        8.4807337678923566671e-02, +3.9530981588194673068e-16,
        8.5011243820190429688e-02, +0,
        8.3074323040850828193e-02, -4.3022503210464894539e-17,
        8.3265960216522216797e-02, +0,
        8.1369880712663267275e-02, -6.3063867569127169744e-18,
        8.1549942493438720703e-02, +0,
        7.9692445771216036121e-02, -5.0787623072962671502e-17,
        7.9861581325531005859e-02, +0,
        7.8040568735575632786e-02, -3.8810063021216721741e-16,
        7.8199386596679687500e-02, +0,
        7.6412797391314235540e-02, +4.1246529500495762995e-16,
        7.6561868190765380859e-02, +0,
        7.4807854772808823896e-02, -3.7025599052186724156e-16,
        7.4947714805603027344e-02, +0,
        7.3224639528778112663e-02, +4.2209138483206712401e-17,
        7.3355793952941894531e-02, +0,
        7.1661929761571485642e-02, -3.2074473649855177622e-16,
        7.1784853935241699219e-02, +0,
        7.0118738881148168218e-02, -2.5371257235753296804e-16,
        7.0233881473541259766e-02, +0,
        6.8594137996416115755e-02, +3.3796987842548399135e-16,
        6.8701922893524169922e-02, +0,
        6.7087137393172291411e-02, +5.5061492696328852397e-17,
        6.7187964916229248047e-02, +0,
        6.5596983299946565182e-02, -2.1580863111502565280e-16,
        6.5691232681274414062e-02, +0,
        6.4122802037412718335e-02, -3.1315661827469233434e-16,
        6.4210832118988037109e-02, +0,
        6.2426231582525915087e-02, -2.5758980071296622188e-16,
        6.2507450580596923828e-02, +0,
        6.0781559928021700046e-02, +1.3736899336217710591e-16,
        6.0856521129608154297e-02, +0,
        5.9432882624005145544e-02, +2.2246097394328856474e-16,
        5.9502959251403808594e-02, +0,
        5.8132551274581167888e-02, -6.2525053236379489390e-18,
        5.8198124170303344727e-02, +0,
        5.6876611930681164608e-02, -2.6589930995607417149e-16,
        5.6938022375106811523e-02, +0,
        5.5661522654748551986e-02, -4.2736362859832186197e-16,
```



```
        5.5719077587127685547e-02, +0,
        5.4484124463757943602e-02, -1.6708067365310384253e-16,
        5.4538100957870483398e-02, +0,
        5.3341582449436764080e-02, +3.3271673004611311850e-17,
        5.3392231464385986328e-02, +0,
        5.2231267345892007370e-02, -3.5593396674200571616e-16,
        5.2278816699981689453e-02, +0,
        5.1150874758829623090e-02, +1.4432815841187114832e-16,
        5.1195532083511352539e-02, +0,
        5.0098306612679444072e-02, +9.4680943793589404083e-17,
        5.0140261650085449219e-02, +0,
        4.9071641675614507960e-02, +2.1131168520301896817e-16,
        4.9111068248748779297e-02, +0,
        4.8069135772851545596e-02, +1.6035336741307516296e-16,
        4.8106193542480468750e-02, +0,
        4.7089192241088539959e-02, -2.2491738698796901479e-16,
        4.7124028205871582031e-02, +0,
        4.6130362086062248750e-02, -1.5111423469578965206e-16,
        4.6163119251251220702e-02, +0,
        4.5191314382707403752e-02, +4.1989325207399786612e-16,
        4.5222103595733642578e-02, +0,
        4.4270836390474244126e-02, -4.1432635292331004454e-16,
        4.4299781322479248047e-02, +0,
        4.3367774164955186222e-02, -3.0615383054587355892e-16,
        4.3394982814788818359e-02, +0,
        4.2481121875321825598e-02, -3.6730166956273555173e-16,
        4.2506694793701171875e-02, +0,
        4.1609902899457651415e-02, -4.4226425958068821782e-16,
        4.1633933782577514648e-02, +0,
        4.0753259129372665370e-02, +1.9801161516527046872e-16,
        4.0775835514068603516e-02, +0,
        3.9910361780060910064e-02, +8.2560620036613164573e-18,
        3.9931565523147583008e-02, +0,
        3.9080441183869218946e-02, +3.9908991939242971628e-17,
        3.9100348949432373047e-02, +0,
        3.8262816593271686827e-02, +9.5182237812195590276e-17,
        3.8281500339508056641e-02, +0,
        3.7456806948784837630e-02, +1.5213508760679563439e-16,
        3.7474334239959716797e-02, +0,
        3.6661849947035918262e-02, +7.3335516005184616486e-17,
        3.6678284406661987305e-02, +0,
        3.5877353272533163420e-02, -1.3007348019891714540e-16,
        3.5892754793167114258e-02, +0,
        3.5102754135096780885e-02, -2.9903662298950558656e-16,
        3.5117179155349731445e-02, +0,
        3.4337638360670830195e-02, +2.9656295131966114331e-16,
        3.4351140260696411133e-02, +0,
        3.3581472523789734907e-02, +3.4810947205572817820e-16,
        3.3594101667404174805e-02, +0,
        3.2833871859357266487e-02, -3.8885440174405159838e-16,
        3.2845675945281982422e-02, +0,
        3.2094421679560447558e-02, +5.8805134853032009978e-17,
        3.2105445861816406250e-02, +0,
        3.1243584858944295490e-02, +2.8737383773884313066e-17,
        3.1253755092620849609e-02, +0,
        0, 0, 0, 0
};

static const double C[] = {
        0.0,
        0.125,
        1.2980742146337069071e+33,
        7.8539816339744827900e-01,
        1.5707963267948965580e+00,
        6.1232339957367658860e-17,
        -3.1415926535897931160e+00,
```

```
        -1.2246467991473531772e-16,
        -3.3333333333332757189333178635417910107486063300 9e-0001,
        +1.9999999994267162423008649761039472181743863137 9e-0001,
        -1.4285696556542863689618301332472720598048415835 6e-0001,
        +1.1089498149631708140510771847504016808416482564 1e-0001,
};

#define zero            C[0]
#define twom3           C[1]
#define two110          C[2]
#define pio4            C[3]
#define pio2            C[4]
#define pio2_lo         C[5]
#define mpi             C[6]
#define mpi_lo          C[7]
#define p1              C[8]
#define p2              C[9]
#define p3              C[10]
#define p4              C[11]

double
atan2(double oy, double ox)
{
 21 atan2(double oy, double ox) {
        double ah, al, t, xh, x, y, z;
        int i, k, hx, hy, sx, sy;

#endif /* ! codereview */
#ifndef lint
        volatile int inexact __unused;
#endif

        hy = ((int *)&oy)[HIWORD];
        sy = hy & 0x80000000;
        hy &= ~0x80000000;

        hx = ((int *)&ox)[HIWORD];
        sx = hx & 0x80000000;
        hx &= ~0x80000000;

        if (hy > hx || (hy == hx && ((unsigned *)&oy)[LOWORD] >
            ((unsigned *)&ox)[LOWORD])) {
                i = hx;
                hx = hy;
                hy = i;
                x = fabs(oy);
                y = fabs(ox);

#endif /* ! codereview */
                if (sx) {
                        ah = pio2;
                        al = pio2_lo;
                } else {
                        ah = -pio2;
                        al = -pio2_lo;
                        sy ^= 0x80000000;
                }
        } else {
                x = fabs(ox);
                y = fabs(oy);

#endif /* ! codereview */
                if (sx) {
                        ah = mpi;
                        al = mpi_lo;
                        sy ^= 0x80000000;
```



```
                } else {
                        ah = al = zero;
                }
        }

        if (hx >= 0x7fe00000 || hx - hy >= 0x03600000) {
                if (hx >= 0x7ff00000) {
                        if (((hx ^ 0x7ff00000) | ((int *)&x)[LOWORD]) != 0)
                                return (ox * oy);

#endif /* ! codereview */
                        if (hy >= 0x7ff00000)
                                ah += pio4;

#endif /* ! codereview */
#ifndef lint
                        inexact = (int)ah;      /* inexact if ah != 0 */
#endif
                        return ((sy) ? -ah : ah);
 24                     return ((sy)? -ah : ah);
                }

#endif /* ! codereview */
                if (hx - hy >= 0x03600000) {
                        if ((int)ah == 0)
                                ah = y / x;

                        return ((sy) ? -ah : ah);
 26                     return ((sy)? -ah : ah);
                }

#endif /* ! codereview */
                y *= twom3;
                x *= twom3;
                hy -= 0x00300000;
                hx -= 0x00300000;
        } else if (hy < 0x00100000) {
                if ((hy | ((int *)&y)[LOWORD]) == 0) {
                        if ((hx | ((int *)&x)[LOWORD]) == 0)
                                return (_SVID_libm_err(ox, oy, 3));

#endif /* ! codereview */
#ifndef lint
                        inexact = (int)ah;      /* inexact if ah != 0 */
#endif
                        return ((sy) ? -ah : ah);
 28                     return ((sy)? -ah : ah);
                }

#endif /* ! codereview */
                y *= two110;
                x *= two110;
                hy = ((int *)&y)[HIWORD];
                hx = ((int *)&x)[HIWORD];
        }

        k = (((hx - hy) + 0x00004000) >> 13) & ~0x3;

#endif /* ! codereview */
        if (k > 644)
                k = 644;

#endif /* ! codereview */
        ah += TBL[k];
        al += TBL[k + 1];
        t = TBL[k + 2];
```

```
        al += TBL[k+1];
        t = TBL[k+2];

        xh = x;
        ((int *)&xh)[LOWORD] = 0;
        z = ((y - t * xh) - t * (x - xh)) / (x + y * t);
        x = z * z;
        t = ah + (z + (al + (z * x) * (p1 + x * (p2 + x * (p3 + x * p4)))));
        return ((sy) ? -t : t);
        return ((sy)? -t : t);
}
```

_____***unchanged_portion_omitted_***

```
**********************************************************
    1351 Thu Jun  6 20:37:40 2019
new/usr/src/lib/libm/common/C/atan2pi.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

#endif /* ! codereview */
/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2005 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

/*
 * atan2pi(x) = atan2(x)/pi
 */

#include "libm.h"

static const double invpi = 0.31830988618379067153776 75;

double
atan2pi(double y, double x)
{
atan2pi(double y, double x) {
        int ix, iy;

        if (x == 0.0 && y == 0.0) {
                ix = ((int *)&x)[HIWORD];
                iy = ((int *)&y)[HIWORD];

#endif /* ! codereview */
                if (ix >= 0)
                        return (y);

                return ((iy >= 0) ? 1.0 : -1.0);
                return ((iy >= 0)? 1.0 : -1.0);
        }

#endif /* ! codereview */
        return (atan2(y, x) * invpi);
}
```

```
**********************************************************
    2034 Thu Jun  6 20:37:41 2019
new/usr/src/lib/libm/common/C/atanh.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __atanh = atanh


/* INDENT OFF */
/*
 * atanh(x)
 * Code originated from 4.3bsd.
 * Modified by K.C. Ng for SUN 4.0 libm.
 * Method :
 *                  1              2x                          x
 *      atanh(x) = --- * log(1 + -------) = 0.5 * log1p(2 * --------)
 *                  2             1 - x                       1 - x
 * Note: to guarantee atanh(-x) = -atanh(x), we use
 *                 sign(x)             |x|
 *      atanh(x) = ------- * log1p(2*-------).
 *                    2              1 - |x|
 *
 * Special cases:
 *      atanh(x) is NaN if |x| > 1 with signal;
 *      atanh(NaN) is that NaN with no signal;
 *      atanh(+-1) is +-INF with signal.
 */
/* INDENT ON */

#include "libm.h"
#include "libm_protos.h"
#include <math.h>

double
atanh(double x)
```



```
{
atanh(double x) {
        double t;

        if (isnan(x))
                return (x * x);         /* switched from x + x for Cheetah */

#endif /* ! codereview */
        t = fabs(x);

#endif /* ! codereview */
        if (t > 1.0)
                return (_SVID_libm_err(x, x, 30));      /* sNaN */

#endif /* ! codereview */
        if (t == 1.0)
                return (_SVID_libm_err(x, x, 31));      /* x/0; */

#endif /* ! codereview */
        t = t / (1.0 - t);
        return (copysign(0.5, x) * log1p(t + t));
}
```

```
**********************************************************
   13685 Thu Jun  6 20:37:41 2019
new/usr/src/lib/libm/common/C/cbrt.c
11210 libm should be cstyle(10NBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#include "libm_macros.h"

/* INDENT OFF */

/*
 *  cbrt: double precision cube root
 *
 *  Algorithm: bit hacking, table lookup, and polynomial approximation
 *
 *  For normal x, write x = s*2^(3j)*z where s = +/-1, j is an integer,
 *  and 1 <= z < 8.  Let y := s*2^j.  From a table, find u such that
 *  u^3 is computable exactly and |(z-u^3)/u^3| <~ 2^-8.  We construct
 *  y, z, and the table index from x by a few integer operations.
 *
 *  Now cbrt(x) = y*u*(1+t)^(1/3) where t = (z-u^3)/u^3.  We approximate
 *  (1+t)^(1/3) by a polynomial 1+p(t), where p(t) := t*(p1+t*(p2+...+
 *  (p5+t*p6))).  By computing the result as y*(u+u*p(t)), we can bound
 *  the worst case error by .51 ulp.
 *
 *  Notes:
 *
 *  1. For subnormal x, we scale x by 2^54, compute the cube root, and
 *     scale the result by 2^-18.
 *
 *  2. cbrt(+/-inf) = +/-inf and cbrt(NaN) is NaN.
 */

/*
 * for i = 0, ..., 385
 *   form x(i) with high word 0x3ff00000 + (i << 13) and low word 0;
```



```
 *   then TBL[i] = cbrt(x(i)) rounded to 17 significant bits
 */
static const double __libm_TBL_cbrt[] = {
        1.00000000000000000e+00, 1.00259399414062500e+00,
        1.00518798828125000e+00, 1.00775146484375000e+00,
        1.01031494140625000e+00, 1.01284790039062500e+00,
        1.01538085937500000e+00, 1.01791381835937500e+00,
        1.02041625976562500e+00, 1.02290344238281250e+00,
        1.02539062500000000e+00, 1.02786254882812500e+00,
        1.03031921386718750e+00, 1.03277587890625000e+00,
        1.03520202636718750e+00, 1.03762817382812500e+00,
        1.04003906250000000e+00, 1.04244995117187500e+00,
        1.04483032226562500e+00, 1.04721069335937500e+00,
        1.04959106445312500e+00, 1.05194091796875000e+00,
        1.05429077148437500e+00, 1.05662536621093750e+00,
        1.05895996093750000e+00, 1.06127929687500000e+00,
        1.06358337402343750e+00, 1.06587219238281250e+00,
        1.06816101074218750e+00, 1.07044982910156250e+00,
        1.07270812988281250e+00, 1.07496643066406250e+00,
        1.07722473144531250e+00, 1.07945251464843750e+00,
        1.08168029785156250e+00, 1.08390808105468750e+00,
        1.08612060546875000e+00, 1.08831787109375000e+00,
        1.09051513671875000e+00, 1.09269714355468750e+00,
        1.09487915039062500e+00, 1.09704589843750000e+00,
        1.09921264648437500e+00, 1.10136413574218750e+00,
        1.10350036621093750e+00, 1.10563659667968750e+00,
        1.10775756835937500e+00, 1.10987854003906250e+00,
        1.11198425292968750e+00, 1.11408996582031250e+00,
        1.11618041992187500e+00, 1.11827087402343750e+00,
        1.12034606933593750e+00, 1.12242126464843750e+00,
        1.12448120117187500e+00, 1.12654113769531250e+00,
        1.12858581542968750e+00, 1.13063049316406250e+00,
        1.13265991210937500e+00, 1.13468933105468750e+00,
        1.13670349121093750e+00, 1.13871765136718750e+00,
        1.14073181152343750e+00, 1.14273071289062500e+00,
        1.14471435546875000e+00, 1.14669799804687500e+00,
        1.14868164062500000e+00, 1.15065002441406250e+00,
        1.15260314941406250e+00, 1.15457153320312500e+00,
        1.15650939941406250e+00, 1.15846252441406250e+00,
        1.16040039062500000e+00, 1.16232299804687500e+00,
        1.16424560546875000e+00, 1.16616821289062500e+00,
        1.16807556152343750e+00, 1.16998291015625000e+00,
        1.17189025878906250e+00, 1.17378234863281250e+00,
        1.17567443847656250e+00, 1.17755126953125000e+00,
        1.17942810058593750e+00, 1.18128967285156250e+00,
        1.18315124511718750e+00, 1.18501281738281250e+00,
        1.18685913085937500e+00, 1.18870544433593750e+00,
        1.19055175781250000e+00, 1.19238281250000000e+00,
        1.19421386718750000e+00, 1.19602966308593750e+00,
        1.19786071777343750e+00, 1.19966125488281250e+00,
        1.20147705078125000e+00, 1.20327758789062500e+00,
        1.20507812500000000e+00, 1.20686340332031250e+00,
        1.20864868164062500e+00, 1.21043395996093750e+00,
        1.21220397949218750e+00, 1.21397399902343750e+00,
        1.21572875976562500e+00, 1.21749877929687500e+00,
        1.21925354003906250e+00, 1.22099304199218750e+00,
        1.22274780273437500e+00, 1.22448730468750000e+00,
        1.22621154785156250e+00, 1.22795104980468750e+00,
        1.22967529296875000e+00, 1.23138427734375000e+00,
        1.23310852050781250e+00, 1.23481750488281250e+00,
        1.23652648925781250e+00, 1.23822021484375000e+00,
        1.23991394042968750e+00, 1.24160766601562500e+00,
        1.24330139160156250e+00, 1.24497985839843750e+00,
        1.24665832519531250e+00, 1.24833679199218750e+00,
        1.25000000000000000e+00, 1.25166320800781250e+00,
        1.25332641601562500e+00, 1.25497436523437500e+00,
```

```
        1.25663757324218750e+00, 1.25828552246093750e+00,
        1.25991821289062500e+00, 1.26319885253906250e+00,
        1.26644897460937500e+00, 1.26968383789062500e+00,
        1.27290344238281250e+00, 1.27612304687500000e+00,
        1.27931213378906250e+00, 1.28248596191406250e+00,
        1.28564453125000000e+00, 1.28878784179687500e+00,
        1.29191589355468750e+00, 1.29502868652343750e+00,
        1.29812622070312500e+00, 1.30120849609375000e+00,
        1.30427551269531250e+00, 1.30732727050781250e+00,
        1.31036376953125000e+00, 1.31340026855468750e+00,
        1.31640625000000000e+00, 1.31941223144531250e+00,
        1.32238769531250000e+00, 1.32536315917968750e+00,
        1.32832336425781250e+00, 1.33126831054687500e+00,
        1.33419799804687500e+00, 1.33712768554687500e+00,
        1.34002685546875000e+00, 1.34292602539062500e+00,
        1.34580993652343750e+00, 1.34867858886718750e+00,
        1.35153198242187500e+00, 1.35437011718750000e+00,
        1.35720825195312500e+00, 1.36003112792968750e+00,
        1.36283874511718750e+00, 1.36564636230468750e+00,
        1.36842346191406250e+00, 1.37120056152343750e+00,
        1.37396240234375000e+00, 1.37672424316406250e+00,
        1.37945556640625000e+00, 1.38218688964843750e+00,
        1.38491821289062500e+00, 1.38761901855468750e+00,
        1.39031982421875000e+00, 1.39302062988281250e+00,
        1.39569091796875000e+00, 1.39836120605468750e+00,
        1.40101623535156250e+00, 1.40367126464843750e+00,
        1.40631103515625000e+00, 1.40893554687500000e+00,
        1.41156005859375000e+00, 1.41416931152343750e+00,
        1.41676330566406250e+00, 1.41935729980468750e+00,
        1.42193603515625000e+00, 1.42449951171875000e+00,
        1.42706298828125000e+00, 1.42962646484375000e+00,
        1.43215942382812500e+00, 1.43469238281250000e+00,
        1.43722534179687500e+00, 1.43974304199218750e+00,
        1.44224548339843750e+00, 1.44474792480468750e+00,
        1.44723510742187500e+00, 1.44972229003906250e+00,
        1.45219421386718750e+00, 1.45466613769531250e+00,
        1.45712280273437500e+00, 1.45956420898437500e+00,
        1.46200561523437500e+00, 1.46444702148437500e+00,
        1.46687316894531250e+00, 1.46928405761718750e+00,
        1.47169494628906250e+00, 1.47409057617187500e+00,
        1.47648620605468750e+00, 1.47886657714843750e+00,
        1.48124694824218750e+00, 1.48361206054687500e+00,
        1.48597717285156250e+00, 1.48834228515625000e+00,
        1.49067687988281250e+00, 1.49302673339843750e+00,
        1.49536132812500000e+00, 1.49768066406250000e+00,
        1.50000000000000000e+00, 1.50230407714843750e+00,
        1.50460815429687500e+00, 1.50691223144531250e+00,
        1.50920104980468750e+00, 1.51148986816406250e+00,
        1.51376342773437500e+00, 1.51603698730468750e+00,
        1.51829528808593750e+00, 1.52055358886718750e+00,
        1.52279663085937500e+00, 1.52503967285156250e+00,
        1.52728271484375000e+00, 1.52951049804687500e+00,
        1.53173828125000000e+00, 1.53395080566406250e+00,
        1.53616333007812500e+00, 1.53836059570312500e+00,
        1.54055786132812500e+00, 1.54275512695312500e+00,
        1.54493713378906250e+00, 1.54711914062500000e+00,
        1.54928588867187500e+00, 1.55145263671875000e+00,
        1.55361938476562500e+00, 1.55577087402343750e+00,
        1.55792236328125000e+00, 1.56005859375000000e+00,
        1.56219482421875000e+00, 1.56433105468750000e+00,
        1.56645202636718750e+00, 1.56857299804687500e+00,
        1.57069396972656250e+00, 1.57279968261718750e+00,
        1.57490539550781250e+00, 1.57699584960937500e+00,
        1.57908630371093750e+00, 1.58117675781250000e+00,
        1.58325195312500000e+00, 1.58532714843750000e+00,
        1.58740234375000000e+00, 1.59152221679687500e+00,
```



```
        1.59562683105468750e+00, 1.59970092773437500e+00,
        1.60375976562500000e+00, 1.60780334472656250e+00,
        1.61183166503906250e+00, 1.61582946777343750e+00,
        1.61981201171875000e+00, 1.62376403808593750e+00,
        1.62770080566406250e+00, 1.63162231445312500e+00,
        1.63552856445312500e+00, 1.63941955566406250e+00,
        1.64328002929687500e+00, 1.64714050292968750e+00,
        1.65097045898437500e+00, 1.65476989746093750e+00,
        1.65856933593750000e+00, 1.66235351562500000e+00,
        1.66610717773437500e+00, 1.66986083984375000e+00,
        1.67358398437500000e+00, 1.67729187011718750e+00,
        1.68098449707031250e+00, 1.68466186523437500e+00,
        1.68832397460937500e+00, 1.69197082519531250e+00,
        1.69560241699218750e+00, 1.69921875000000000e+00,
        1.70281982421875000e+00, 1.70640563964843750e+00,
        1.70997619628906250e+00, 1.71353149414062500e+00,
        1.71707153320312500e+00, 1.72059631347656250e+00,
        1.72410583496093750e+00, 1.72760009765625000e+00,
        1.73109436035156250e+00, 1.73455810546875000e+00,
        1.73800659179687500e+00, 1.74145507812500000e+00,
        1.74488830566406250e+00, 1.74829101562500000e+00,
        1.75169372558593750e+00, 1.75508117675781250e+00,
        1.75846862792968750e+00, 1.76182556152343750e+00,
        1.76516723632812500e+00, 1.76850891113281250e+00,
        1.77183532714843750e+00, 1.77514648437500000e+00,
        1.77844238281250000e+00, 1.78173828125000000e+00,
        1.78500366210937500e+00, 1.78826904296875000e+00,
        1.79151916503906250e+00, 1.79476928710937500e+00,
        1.79798889160156250e+00, 1.80120849609375000e+00,
        1.80441284179687500e+00, 1.80760192871093750e+00,
        1.81079101562500000e+00, 1.81396484375000000e+00,
        1.81712341308593750e+00, 1.82026672363281250e+00,
        1.82341003417968750e+00, 1.82653808593750000e+00,
        1.82965087890625000e+00, 1.83276367187500000e+00,
        1.83586120605468750e+00, 1.83894348144531250e+00,
        1.84201049804687500e+00, 1.84507751464843750e+00,
        1.84812927246093750e+00, 1.85118103027343750e+00,
        1.85421752929687500e+00, 1.85723876953125000e+00,
        1.86026000976562500e+00, 1.86326599121093750e+00,
        1.86625671386718750e+00, 1.86924743652343750e+00,
        1.87222290039062500e+00, 1.87518310546875000e+00,
        1.87814331054687500e+00, 1.88108825683593750e+00,
        1.88403320312500000e+00, 1.88696289062500000e+00,
        1.88987731933593750e+00, 1.89279174804687500e+00,
        1.89569091796875000e+00, 1.89859008789062500e+00,
        1.90147399902343750e+00, 1.90435791015625000e+00,
        1.90722656250000000e+00, 1.91007995605468750e+00,
        1.91293334960937500e+00, 1.91577148437500000e+00,
        1.91860961914062500e+00, 1.92143249511718750e+00,
        1.92425537109375000e+00, 1.92706298828125000e+00,
        1.92985534667968750e+00, 1.93264770507812500e+00,
        1.93544006347656250e+00, 1.93821716308593750e+00,
        1.94097900390625000e+00, 1.94374084472656250e+00,
        1.94650268554687500e+00, 1.94924926757812500e+00,
        1.95198059082031250e+00, 1.95471191406250000e+00,
        1.95742797851562500e+00, 1.96014404296875000e+00,
        1.96286010742187500e+00, 1.96556091308593750e+00,
        1.96824645996093750e+00, 1.97093200683593750e+00,
        1.97361755371093750e+00, 1.97628784179687500e+00,
        1.97894287109375000e+00, 1.98159790039062500e+00,
        1.98425292968750000e+00, 1.98689270019531250e+00,
        1.98953247070312500e+00, 1.99215698242187500e+00,
        1.99478149414062500e+00, 1.99739074707031250e+00,
  1.00000000000000000e+00, 1.00259399414062500e+00, 1.00518798828125000e+00,
  1.00775146484375000e+00, 1.01031494140625000e+00, 1.01284790039062500e+00,
  1.01538085937500000e+00, 1.01791381835937500e+00, 1.02041625976562500e+00,
```

```
1.02290344238281250e+00, 1.02539062500000000e+00, 1.02786254882812500e+00,
1.03031921386718750e+00, 1.03277587890625000e+00, 1.03520202636718750e+00,
1.03762817382812500e+00, 1.04003906250000000e+00, 1.04244995117187500e+00,
1.04483032226562500e+00, 1.04721069335937500e+00, 1.04959106445312500e+00,
1.05194091796875000e+00, 1.05429077148437500e+00, 1.05662536621093750e+00,
1.05895996093750000e+00, 1.06127929687500000e+00, 1.06358337402343750e+00,
1.06587219238281250e+00, 1.06816101074218750e+00, 1.07044982910156250e+00,
1.07270812988281250e+00, 1.07496643066406250e+00, 1.07722473144531250e+00,
1.07945251464843750e+00, 1.08168029785156250e+00, 1.08390808105468750e+00,
1.08612060546875000e+00, 1.08831787109375000e+00, 1.09051513671875000e+00,
1.09269714355468750e+00, 1.09487915039062500e+00, 1.09704589843750000e+00,
1.09921264648437500e+00, 1.10136413574218750e+00, 1.10350036621093750e+00,
1.10563659667968750e+00, 1.10775756835937500e+00, 1.10987854003906250e+00,
1.11198425292968750e+00, 1.11408996582031250e+00, 1.11618041992187500e+00,
1.11827087402343750e+00, 1.12034606933593750e+00, 1.12242126464843750e+00,
1.12448120117187500e+00, 1.12654113769531250e+00, 1.12858581542968750e+00,
1.13063049316406250e+00, 1.13265991210937500e+00, 1.13468933105468750e+00,
1.13670349121093750e+00, 1.13871765136718750e+00, 1.14073181152343750e+00,
1.14273071289062500e+00, 1.14471435546875000e+00, 1.14669799804687500e+00,
1.14868164062500000e+00, 1.15065002441406250e+00, 1.15260314941406250e+00,
1.15457153320312500e+00, 1.15650939941406250e+00, 1.15846252441406250e+00,
1.16040039062500000e+00, 1.16232299804687500e+00, 1.16424560546875000e+00,
1.16616821289062500e+00, 1.16807556152343750e+00, 1.16998291015625000e+00,
1.17189025878906250e+00, 1.17378234863281250e+00, 1.17567443847656250e+00,
1.17755126953125000e+00, 1.17942810058593750e+00, 1.18128967285156250e+00,
1.18315124511718750e+00, 1.18501281738281250e+00, 1.18685913085937500e+00,
1.18870544433593750e+00, 1.19055175781250000e+00, 1.19238281250000000e+00,
1.19421386718750000e+00, 1.19602966308593750e+00, 1.19786071777343750e+00,
1.19966125488281250e+00, 1.20147705078125000e+00, 1.20327758789062500e+00,
1.20507812500000000e+00, 1.20686340332031250e+00, 1.20864868164062500e+00,
1.21043395996093750e+00, 1.21220397949218750e+00, 1.21397399902343750e+00,
1.21572875976562500e+00, 1.21749877929687500e+00, 1.21925354003906250e+00,
1.22099304199218750e+00, 1.22274780273437500e+00, 1.22448730468750000e+00,
1.22621154785156250e+00, 1.22795104980468750e+00, 1.22967529296875000e+00,
1.23138427734375000e+00, 1.23310852050781250e+00, 1.23481750488281250e+00,
1.23652648925781250e+00, 1.23822021484375000e+00, 1.23991394042968750e+00,
1.24160766601562500e+00, 1.24330139160156250e+00, 1.24497985839843750e+00,
1.24665832519531250e+00, 1.24833679199218750e+00, 1.25000000000000000e+00,
1.25166320800781250e+00, 1.25332641601562500e+00, 1.25497436523437500e+00,
1.25663757324218750e+00, 1.25828552246093750e+00, 1.25991821289062500e+00,
1.26319885253906250e+00, 1.26644897460937500e+00, 1.26968383789062500e+00,
1.27290344238281250e+00, 1.27612304687500000e+00, 1.27931213378906250e+00,
1.28248596191406250e+00, 1.28564453125000000e+00, 1.28878781679687500e+00,
1.29191589355468750e+00, 1.29502868652343750e+00, 1.29812622070312500e+00,
1.30120849609375000e+00, 1.30427551269531250e+00, 1.30732727050781250e+00,
1.31036376953125000e+00, 1.31340026855468750e+00, 1.31640625000000000e+00,
1.31941223144531250e+00, 1.32238769531250000e+00, 1.32536315917968750e+00,
1.32832336425781250e+00, 1.33126831054687500e+00, 1.33419799804687500e+00,
1.33712768554687500e+00, 1.34002685546875000e+00, 1.34292602539062500e+00,
1.34580993652343750e+00, 1.34867858886718750e+00, 1.35153198242187500e+00,
1.35437011718750000e+00, 1.35720825195312500e+00, 1.36003112792968750e+00,
1.36283874511718750e+00, 1.36564636230468750e+00, 1.36842346191406250e+00,
1.37120056152343750e+00, 1.37396240234375000e+00, 1.37672424316406250e+00,
1.37945556640625000e+00, 1.38218688964843750e+00, 1.38491821289062500e+00,
1.38761901855468750e+00, 1.39031982421875000e+00, 1.39302062988281250e+00,
1.39569091796875000e+00, 1.39836120605468750e+00, 1.40101623535156250e+00,
1.40367126464843750e+00, 1.40631103515625000e+00, 1.40893554687500000e+00,
1.41156005859375000e+00, 1.41416931152343750e+00, 1.41676330566406250e+00,
1.41935729980468750e+00, 1.42193603515625000e+00, 1.42449951171875000e+00,
1.42706298828125000e+00, 1.42962646484375000e+00, 1.43215942382812500e+00,
1.43469238281250000e+00, 1.43722534179687500e+00, 1.43974304199218750e+00,
1.44224548339843750e+00, 1.44474792480468750e+00, 1.44723510742187500e+00,
1.44972229003906250e+00, 1.45219421386718750e+00, 1.45466613769531250e+00,
1.45712280273437500e+00, 1.45956420898437500e+00, 1.46200561523437500e+00,
1.46444702148437500e+00, 1.46687316894531250e+00, 1.46928405761718750e+00,
1.47169494628906250e+00, 1.47409057617187500e+00, 1.47648620605468750e+00,
```



```
1.47886657714843750e+00, 1.48124694824218750e+00, 1.48361206054687500e+00,
1.48597717285156250e+00, 1.48834228515625000e+00, 1.49067687988281250e+00,
1.49302673339843750e+00, 1.49536132812500000e+00, 1.49768066406250000e+00,
1.50000000000000000e+00, 1.50230407714843750e+00, 1.50460815429687500e+00,
1.50691223144531250e+00, 1.50920104980468750e+00, 1.51148986816406250e+00,
1.51376342773437500e+00, 1.51603698730468750e+00, 1.51829528808593750e+00,
1.52055358886718750e+00, 1.52279663085937500e+00, 1.52503967285156250e+00,
1.52728271484375000e+00, 1.52951049804687500e+00, 1.53173828125000000e+00,
1.53395080566406250e+00, 1.53616333007812500e+00, 1.53836059570312500e+00,
1.54055786132812500e+00, 1.54275512695312500e+00, 1.54493713378906250e+00,
1.54711914062500000e+00, 1.54928588867187500e+00, 1.55145263671875000e+00,
1.55361938476562500e+00, 1.55577087402343750e+00, 1.55792236328125000e+00,
1.56005859375000000e+00, 1.56219482421875000e+00, 1.56433105468750000e+00,
1.56645202636718750e+00, 1.56857299804687500e+00, 1.57069396972656250e+00,
1.57279968261718750e+00, 1.57490539550781250e+00, 1.57699584960937500e+00,
1.57908630371093750e+00, 1.58117675781250000e+00, 1.58325195312500000e+00,
1.58532714843750000e+00, 1.58740234375000000e+00, 1.59152221679687500e+00,
1.59562683105468750e+00, 1.59970092773437500e+00, 1.60375976562500000e+00,
1.60780344472656250e+00, 1.61183166503906250e+00, 1.61582946777343750e+00,
1.61981201171875000e+00, 1.62376403808593750e+00, 1.62770080566406250e+00,
1.63162231445312500e+00, 1.63552856445312500e+00, 1.63941955566406250e+00,
1.64328002929687500e+00, 1.64714050292968750e+00, 1.65097045898437500e+00,
1.65476989746093750e+00, 1.65856933593750000e+00, 1.66235351562500000e+00,
1.66610717773437500e+00, 1.66986083984375000e+00, 1.67358398437500000e+00,
1.67729187011718750e+00, 1.68098449707031250e+00, 1.68466186523437500e+00,
1.68832397460937500e+00, 1.69197082519531250e+00, 1.69560241699218750e+00,
1.69921875000000000e+00, 1.70281982421875000e+00, 1.70640563964843750e+00,
1.70997619628906250e+00, 1.71353149414062500e+00, 1.71707153320312500e+00,
1.72059631347656250e+00, 1.72410583496093750e+00, 1.72760009765625000e+00,
1.73109436035156250e+00, 1.73455810546875000e+00, 1.73800659179687500e+00,
1.74145507812500000e+00, 1.74488830566406250e+00, 1.74829101562500000e+00,
1.75169372558593750e+00, 1.75508117675781250e+00, 1.75846862792968750e+00,
1.76182556152343750e+00, 1.76516723632812500e+00, 1.76850891113281250e+00,
1.77183532714843750e+00, 1.77514648437500000e+00, 1.77844238281250000e+00,
1.78173828125000000e+00, 1.78500366210937500e+00, 1.78826904296875000e+00,
1.79151916503906250e+00, 1.79476928710937500e+00, 1.79798889160156250e+00,
1.80120849609375000e+00, 1.80441284179687500e+00, 1.80760192871093750e+00,
1.81079101562500000e+00, 1.81396484375000000e+00, 1.81712341308593750e+00,
1.82026672363281250e+00, 1.82341003417968750e+00, 1.82653808593750000e+00,
1.82965087890625000e+00, 1.83276367187500000e+00, 1.83586120605468750e+00,
1.83894348144531250e+00, 1.84201049804687500e+00, 1.84507751464843750e+00,
1.84812927246093750e+00, 1.85118103027343750e+00, 1.85421752929687500e+00,
1.85723876953125000e+00, 1.86026000976562500e+00, 1.86326599121093750e+00,
1.86625671386718750e+00, 1.86924743652343750e+00, 1.87222290039062500e+00,
1.87518310546875000e+00, 1.87814331054687500e+00, 1.88108825683593750e+00,
1.88403320312500000e+00, 1.88696289062500000e+00, 1.88987319335937500e+00,
1.89279174804687500e+00, 1.89569091796875000e+00, 1.89859008789062500e+00,
1.90147399902343750e+00, 1.90435791015625000e+00, 1.90722656250000000e+00,
1.91007995605468750e+00, 1.91292602539062500e+00, 1.91577148437500000e+00,
1.91860961914062500e+00, 1.92143249511718750e+00, 1.92425537109375000e+00,
1.92706298828125000e+00, 1.92985534667968750e+00, 1.93264770507812500e+00,
1.93544006347656250e+00, 1.93821716308593750e+00, 1.94097900390625000e+00,
1.94374084472656250e+00, 1.94650268554687500e+00, 1.94924926757812500e+00,
1.95198059082031250e+00, 1.95471191406250000e+00, 1.95742797851562500e+00,
1.96014404296875000e+00, 1.96286010742187500e+00, 1.96556091308593750e+00,
1.96824645996093750e+00, 1.97093200683593750e+00, 1.97361755371093750e+00,
1.97628784179687500e+00, 1.97894287109375000e+00, 1.98159790039062500e+00,
1.98425292968750000e+00, 1.98689270019531250e+00, 1.98953247070312500e+00,
1.99215698242187500e+00, 1.99478149414062500e+00, 1.99739074707031250e+00,
        2.00000000000000000e+00,
};
_____unchanged_portion_omitted_

#define p1              C[0]
#define p2              C[1]
#define p3              C[2]
```

```
#define p4              C[3]
#define p5              C[4]
#define p6              C[5]
#define two54           C[6]

/* INDENT ON */

#pragma weak cbrt = __cbrt

double
__cbrt(double x)
double __cbrt(double x)
{
        union {
                unsigned int i[2];
                double d;
        } xx, yy;

#endif /* ! codereview */
        double t, u, w;
        unsigned int hx, sx, ex, j, offset;

        xx.d = x;
        hx = xx.i[HIWORD] & ~0x80000000;
        sx = xx.i[HIWORD] & 0x80000000;

        /* handle special cases */
        if (hx >= 0x7ff00000)           /* x is inf or nan */
#if defined(FPADD_TRAPS_INCOMPLETE_ON_NAN)
                return (hx >= 0x7ff80000 ? x : x + x);

                return hx >= 0x7ff80000 ? x : x + x;
        /* assumes sparc-like QNaN */
#else
                return (x + x);
                return x + x;
#endif

        if (hx < 0x00100000) {          /* x is subnormal or zero */
                if ((hx | xx.i[LOWORD]) == 0)
                        return (x);
                        return x;

                /* scale x to normal range */
                xx.d = x * two54;
                hx = xx.i[HIWORD] & ~0x80000000;
                offset = 0x29800000;
        } else {
                offset = 0x2aa00000;
#endif /* ! codereview */
        }
        else
                offset = 0x2aa00000;

        ex = hx & 0x7ff00000;
        j = (ex >> 2) + (ex >> 4) + (ex >> 6);
        j = j + (j >> 6);
        j = 0x7ff00000 & (j + 0x2aa00); /* j is ex/3 */
        hx -= (j + j + j);
        xx.i[HIWORD] = 0x3ff00000 + hx;

        u = __libm_TBL_cbrt[(hx + 0x1000) >> 13];
        w = u * u * u;
        t = (xx.d - w) / w;

        yy.i[HIWORD] = sx | (j + offset);
```



```
        yy.i[LOWORD] = 0;

        w = t * t;
        return (yy.d * (u + u * (t * (p1 + t * p2 + w * p3) + (w * w) *
            (p4 + t * p5 + w * p6))));
        return yy.d * (u + u * (t * (p1 + t * p2 + w * p3) +
                (w * w) * (p4 + t * p5 + w * p6)));
}
```

_____*unchanged_portion_omitted_*

```
**********************************************************
    1731 Thu Jun  6 20:37:42 2019
new/usr/src/lib/libm/common/C/ceil.c
11210 libm should be cstyle(10NBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

#endif /* ! codereview */
/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2005 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __ceil = ceil

/*
 * ceil(x) returns the least integral value bigger than or equal to x.
 * NOTE: ceil(x) returns result with the same sign as x's, including 0.
 *
 * Modified 8/4/04 for performance.
 */

#include "libm.h"

static const double zero = 0.0, one = 1.0, two52 = 4503599627370496.0;
static const double
        zero = 0.0,
        one = 1.0,
        two52 = 4503599627370496.0;

double
ceil(double x)
{
ceil(double x) {
        double t, w;
        int hx, lx, ix;

        hx = ((int *)&x)[HIWORD];
        lx = ((int *)&x)[LOWORD];
        ix = hx & ~0x80000000;

#endif /* ! codereview */
```



```
        if (ix >= 0x43300000)   /* return x if |x| >= 2^52, or x is NaN */
                return (x * one);

        t = (hx >= 0) ? two52 : -two52;
        t = (hx >= 0)? two52 : -two52;
        w = x + t;
        t = w - t;

#endif /* ! codereview */
        if (ix < 0x3ff00000) {
                if ((ix | lx) == 0)
                        return (x);
                else
                        return ((hx < 0) ? -zero : one);
                        return ((hx < 0)? -zero : one);
        }

        return ((t >= x) ? t : t + one);
        return ((t >= x)? t : t + one);
}
```

_____unchanged_portion_omitted_

```
**********************************************************
    1200 Thu Jun  6 20:37:42 2019
new/usr/src/lib/libm/common/C/copysign.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __copysign = copysign

#include "libm.h"

double
copysign(double x, double y)
{
copysign(double x, double y) {
        int hx, hy;

        hx = ((int *)&x)[HIWORD];
        hy = ((int *)&y)[HIWORD];
        return ((hx ^ hy) >= 0 ? (x) : (-x));
        hx = ((int *) &x)[HIWORD];
        hy = ((int *) &y)[HIWORD];
        return (hx ^ hy) >= 0 ?  (x) : (-x);
}
______unchanged_portion_omitted_
```

```
**********************************************************
    6109 Thu Jun  6 20:37:43 2019
new/usr/src/lib/libm/common/C/cos.c
11210 libm should be cstyle(10NBLD) clean
**********************************************************
   1 /*
   2  * CDDL HEADER START
   3  *
   4  * The contents of this file are subject to the terms of the
   5  * Common Development and Distribution License (the "License").
   6  * You may not use this file except in compliance with the License.
   7  *
   8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
   9  * or http://www.opensolaris.org/os/licensing.
  10  * See the License for the specific language governing permissions
  11  * and limitations under the License.
  12  *
  13  * When distributing Covered Code, include this CDDL HEADER in each
  14  * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  15  * If applicable, add the following below this CDDL HEADER, with the
  16  * fields enclosed by brackets "[]" replaced with your own identifying
  17  * information: Portions Copyright [yyyy] [name of copyright owner]
  18  *
  19  * CDDL HEADER END
  20  */

  22 #endif /* ! codereview */
  23 /*
  24  * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
  25  */

  27 #endif /* ! codereview */
  28 /*
  29  * Copyright 2005 Sun Microsystems, Inc.  All rights reserved.
  30  * Use is subject to license terms.
  31  */

  33 #pragma weak __cos = cos


  21 /* INDENT OFF */
  36 /*
  37  * cos(x)
  38  * Accurate Table look-up algorithm by K.C. Ng, May, 1995.
  39  *
  40  * Algorithm: see sincos.c
  41  */

  43 #include "libm.h"

  45 static const double sc[] = {
  46 /* ONE  = */
  47         1.0,
  32 /* ONE  = */  1.0,
  48 /* NONE = */ -1.0,

  50 #endif /* ! codereview */
  51 /*
  52  * |sin(x) - (x+pp1*x^3+pp2*x^5)| <= 2^-58.79 for |x| < 0.008
  53  */
  54 /* PP1  = */-0.16666666666631655886725205237888952148062785868305 5567,
  55 /* PP2  = */.008333315652997472323564894248466758248475374977974017927,

  34 /* PP1  = */ -0.16666666666631655886725205237888952148062785868305 5567,
  35 /* PP2  = */   .008333315652997472323564894248466758248475374977974017927,
  57 /*
```



```
  58  * |(sin(x) - (x+p1*x^3+...+p4*x^9)|
  59  * |----------------------------- | <= 2^-57.63 for |x| < 0.1953125
  60  * |                x             |
  61  */
  62 /* P1   = */ -1.6666666666666296698052151389203015896 56e-0001,
  63 /* P2   = */ 8.3333333323909512956839934552803363766 63e-0003,
  64 /* P3   = */ -1.9841262379979766927915517782300984039 60e-0004,
  65 /* P4   = */ 2.7534036248542772376499876228483303511 10e-0006,

  67 #endif /* ! codereview */
  68 /*
  69  * |cos(x) - (1+qq1*x^2+qq2*x^4)| <= 2^-55.99 for |x| <= 0.008 (0x3f80624d)
  70  */
  71 /* QQ1  = */-0.49999999999754923818429119819484185427427 29,
  72 /* QQ2  = */0.04166654290435205929454520915835764039877 1740,
  45 /* QQ1  = */ -0.49999999999754923818429119819484185427427 29,
  46 /* QQ2  = */  0.04166654290435205929454520915835764039877 1740,
  73 /* Q1   = */ -0.5,
  74 /* Q2   = */ 4.1666666665003507036809455208607486174 45e-0002,
  75 /* Q3   = */ -1.3888885964369722106942662905778486960 06e-0003,
  76 /* Q4   = */ 2.4785630788585894736795195178929534921 92e-0005,
  77 /* PIO2_H    = */ 1.5707963267489655799 9,
  78 /* PIO2_L    = */ 6.1232339957367658861 30e-17,
  79 /* PIO2_L0   = */ 6.1232339957279221655 64e-17,
  80 /* PIO2_L1   = */ 8.8437205661357011202 55e-29,
  81 /* PI3O2_H   = */ 4.7123889803846896739 97,
  82 /* PI3O2_L   = */ 1.8369701987210297658 39e-16,
  83 /* PI3O2_L0  = */ 1.8369701987203961335 87e-16,
  84 /* PI3O2_L1  = */ 6.3363225247492011422 26e-29,
  85 /* PI5O2_H   = */ 7.8539816339744827899 95,
  86 /* PI5O2_L   = */ 3.0616169978683829430 65e-16,
  87 /* PI5O2_L0  = */ 3.0616169978619415988 65e-16,
  88 /* PI5O2_L1  = */ 6.4413442004336407819 82e-28,
  89 };

  64 /* INDENT ON */

  92 #define ONE             sc[0]
  93 #define PP1             sc[2]
  94 #define PP2             sc[3]
  95 #define P1              sc[4]
  96 #define P2              sc[5]
  97 #define P3              sc[6]
  98 #define P4              sc[7]
  99 #define QQ1             sc[8]
 100 #define QQ2             sc[9]
 101 #define Q1              sc[10]
 102 #define Q2              sc[11]
 103 #define Q3              sc[12]
 104 #define Q4              sc[13]
 105 #define PIO2_H          sc[14]
 106 #define PIO2_L          sc[15]
 107 #define PIO2_L0         sc[16]
 108 #define PIO2_L1         sc[17]
 109 #define PI3O2_H         sc[18]
 110 #define PI3O2_L         sc[19]
 111 #define PI3O2_L0        sc[20]
 112 #define PI3O2_L1        sc[21]
 113 #define PI5O2_H         sc[22]
 114 #define PI5O2_L         sc[23]
 115 #define PI5O2_L0        sc[24]
 116 #define PI5O2_L1        sc[25]

 118 extern const double _TBL_sincos[], _TBL_sincosx[];

 120 double
```

```
121 cos(double x)
122 {
 95 cos(double x) {
123         double z, y[2], w, s, v, p, q;
124         int i, j, n, hx, ix, lx;

126         hx = ((int *)&x)[HIWORD];
127         lx = ((int *)&x)[LOWORD];
128         ix = hx & ~0x80000000;

130         if (ix <= 0x3fc50000) {         /* |x| < 10.5/64 = 0.164062500 */
131                 if (ix < 0x3e400000) {  /* |x| < 2**-27 */
132                         if ((int)x == 0)
133                                 return (ONE);
134                 }
```

```
136 #endif /* ! codereview */
137                 z = x * x;

139 #endif /* ! codereview */
140                 if (ix < 0x3f800000)    /* |x| < 0.008 */
141                         w = z * (QQ1 + z * QQ2);
142                 else
143                         w = z * ((Q1 + z * Q2) + (z * z) * (Q3 + z * Q4));

145 #endif /* ! codereview */
146                 return (ONE + w);
147         }

149         /* for 0.164062500 < x < M, */
150         n = ix >> 20;

152 #endif /* ! codereview */
153         if (n < 0x402) {                /* x < 8 */
154                 i = (((ix >> 12) & 0xff) | 0x100) >> (0x401 - n);
155                 j = i - 10;
156                 x = fabs(x);
157                 v = x - _TBL_sincosx[j];

159 #endif /* ! codereview */
160                 if (((j - 81) ^ (j - 101)) < 0) {
161                         /* near pi/2, cos(pi/2-x)=sin(x) */
162                         p = PIO2_H - x;
163                         i = ix - 0x3ff921fb;
164                         x = p + PIO2_L;

166 #endif /* ! codereview */
167                         if ((i | ((lx - 0x54442D00) & 0xffffff00)) == 0) {
168                                 /* very close to pi/2 */
169                                 x = p + PIO2_L0;
170                                 return (x + PIO2_L1);
171                         }

173 #endif /* ! codereview */
174                         z = x * x;

176 #endif /* ! codereview */
177                         if (((ix - 0x3ff92000) >> 12) == 0) {
178                                 /* |pi/2-x|<2**-8 */
179                                 w = PIO2_L + (z * x) * (PP1 + z * PP2);
180                         } else {
181                                 w = PIO2_L + (z * x) * ((P1 + z * P2) + (z *
182                                     z) * (P3 + z * P4));
108                                 w = PIO2_L + (z * x) * ((P1 + z * P2) +
109                                     (z * z) * (P3 + z * P4));
183                         }
```



```
185 #endif /* ! codereview */
186                         return (p + w);
187                 }

189 #endif /* ! codereview */
190                 s = v * v;

192 #endif /* ! codereview */
193                 if (((j - 282) ^ (j - 302)) < 0) {
194                         /* near 3/2pi, cos(x-3/2pi)=sin(x) */
195                         p = x - PI3O2_H;
196                         i = ix - 0x4012D97C;
197                         x = p - PI3O2_L;

199 #endif /* ! codereview */
200                         if ((i | ((lx - 0x7f332100) & 0xffffff00)) == 0) {
201                                 /* very close to 3/2pi */
202                                 x = p - PI3O2_L0;
203                                 return (x - PI3O2_L1);
204                         }

206 #endif /* ! codereview */
207                         z = x * x;

209 #endif /* ! codereview */
210                         if (((ix - 0x4012D800) >> 9) == 0) {
211                                 /* |x-3/2pi|<2**-8 */
212                                 w = (z * x) * (PP1 + z * PP2) - PI3O2_L;
213                         } else {
214                                 w = (z * x) * ((P1 + z * P2) + (z * z) * (P3 +
215                                     z * P4)) - PI3O2_L;
111                                 w = (z * x) * ((P1 + z * P2) + (z * z)
112                                     * (P3 + z * P4)) - PI3O2_L;
216                         }

218 #endif /* ! codereview */
219                         return (p + w);
220                 }

222 #endif /* ! codereview */
223                 if (((j - 483) ^ (j - 503)) < 0) {
224                         /* near 5pi/2, cos(5pi/2-x)=sin(x) */
225                         p = PI5O2_H - x;
226                         i = ix - 0x401F6A7A;
227                         x = p + PI5O2_L;

229 #endif /* ! codereview */
230                         if ((i | ((lx - 0x29553800) & 0xffffff00)) == 0) {
231                                 /* very close to pi/2 */
232                                 x = p + PI5O2_L0;
233                                 return (x + PI5O2_L1);
234                         }

236 #endif /* ! codereview */
237                         z = x * x;

239 #endif /* ! codereview */
240                         if (((ix - 0x401F6A7A) >> 7) == 0) {
241                                 /* |pi/2-x|<2**-8 */
242                                 w = PI5O2_L + (z * x) * (PP1 + z * PP2);
243                         } else {
244                                 w = PI5O2_L + (z * x) * ((P1 + z * P2) + (z *
245                                     z) * (P3 + z * P4));
114                                 w = PI5O2_L + (z * x) * ((P1 + z * P2) +
115                                     (z * z) * (P3 + z * P4));
```

```
                        }

#endif /* ! codereview */
                        return (p + w);
                }

#endif /* ! codereview */
                j <<= 1;
                w = _TBL_sincos[j];
                z = _TBL_sincos[j + 1];
                z = _TBL_sincos[j+1];
                p = v + (v * s) * (PP1 + s * PP2);
                q = s * (QQ1 + s * QQ2);
                return (z - (w * p - z * q));
        }

        if (ix >= 0x7ff00000)           /* cos(Inf or NaN) is NaN */
                return (x / x);

        /* argument reduction needed */
        n = __rem_pio2(x, y);

#endif /* ! codereview */
        switch (n & 3) {
        case 0:
                return (__k_cos(y[0], y[1]));
        case 1:
                return (-__k_sin(y[0], y[1]));
        case 2:
                return (-__k_cos(y[0], y[1]));
        default:
                return (__k_sin(y[0], y[1]));
        }
}
```

```
**********************************************************
    2426 Thu Jun  6 20:37:43 2019
new/usr/src/lib/libm/common/C/cosh.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

#endif /* ! codereview */
/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2005 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __cosh = cosh


/* INDENT OFF */
/*
 * cosh(x)
 * Code originated from 4.3bsd.
 * Modified by K.C. Ng for SUN 4.0 libm.
 * Method :
 *      1. Replace x by |x| (cosh(x) = cosh(-x)).
 *      2.
 *                                          [ exp(x) - 1 ]^2
 *  0        <= x <= 0.3465  :  cosh(x) := 1 + -------------------
 *                                                                      2*exp(x)
 *
 *                                                 exp(x) +  1/exp(x)
 *  0.3465   <= x <= 22      :  cosh(x) := -------------------
 *                                                                          2
 *  22       <= x <= lnovft  :  cosh(x) := exp(x)/2
 *  lnovft   <= x <  INF     :  cosh(x) := scalbn(exp(x-1024*ln2),1023)
 *
 *      Note: .3465 is a number near one half of ln2.
 *
 * Special cases:
 *      cosh(x) is |x| if x is +INF, -INF, or NaN.
 *      only cosh(0)=1 is exact for finite x.
 */
/* INDENT ON */
```



```
#include "libm.h"

static const double ln2 = 6.93147180559945286227e-01,
static const double
        ln2 = 6.93147180559945286227e-01,
        ln2hi = 6.93147180369123816490e-01,
        ln2lo = 1.90821492927058770002e-10,
        lnovft = 7.09782712893383973096e+02;

double
cosh(double x)
{
cosh(double x) {
        double t, w;

        w = fabs(x);

#endif /* ! codereview */
        if (!finite(w))
                return (w * w);

#endif /* ! codereview */
        if (w < 0.3465) {
                t = expm1(w);
                w = 1.0 + t;

#endif /* ! codereview */
                if (w != 1.0)
                        w = 1.0 + (t * t) / (w + w);

#endif /* ! codereview */
                return (w);
        } else if (w < 22.0) {
                t = exp(w);
                return (0.5 * (t + 1.0 / t));
        } else if (w <= lnovft) {
                return (0.5 * exp(w));
        } else {
                w = (w - 1024 * ln2hi) - 1024 * ln2lo;

#endif /* ! codereview */
                if (w >= ln2)
                        return (_SVID_libm_err(x, x, 5));
                else
                        return (scalbn(exp(w), 1023));
        }
}
```

```
**********************************************************
   13967 Thu Jun  6 20:37:44 2019
new/usr/src/lib/libm/common/C/erf.c
11210 libm should be cstyle(10NBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __erf = erf
#pragma weak __erfc = erfc


/* INDENT OFF */
/*
 * double erf(double x)
 * double erfc(double x)
 *                             x
 *                      2      |\
 *     erf(x)  =  ---------  | exp(-t*t)dt
 *                 sqrt(pi) \|
 *                             0
 *
 *     erfc(x) =  1-erf(x)
 *  Note that
 *              erf(-x) = -erf(x)
 *              erfc(-x) = 2 - erfc(x)
 *
 * Method:
 *      1. For |x| in [0, 0.84375]
 *          erf(x)  = x + x*R(x^2)
 *          erfc(x) = 1 - erf(x)           if x in [-.84375,0.25]
 *                  = 0.5 + ((0.5-x)-x*R)  if x in [0.25,0.84375]
 *         where R = P/Q where P is an odd poly of degree 8 and
 *         Q is an odd poly of degree 10.
 *                                                     -57.90
 *                      | R - (erf(x)-x)/x | <= 2
 *
 *
```



```
 *         Remark. The formula is derived by noting
 *          erf(x) = (2/sqrt(pi))*(x - x^3/3 + x^5/10 - x^7/42 + ....)
 *         and that
 *          2/sqrt(pi) = 1.128379167095512573896158903121545171688
 *         is close to one. The interval is chosen because the fix
 *         point of erf(x) is near 0.6174 (i.e., erf(x)=x when x is
 *         near 0.6174), and by some experiment, 0.84375 is chosen to
 *         guarantee the error is less than one ulp for erf.
 *
 *      2. For |x| in [0.84375,1.25], let s = |x| - 1, and
 *         c = 0.84506291151 rounded to single (24 bits)
 *              erf(x)  = sign(x) * (c  + P1(s)/Q1(s))
 *              erfc(x) = (1-c)  - P1(s)/Q1(s) if x > 0
 *                        1+(c+P1(s)/Q1(s))    if x < 0
 *              |P1/Q1 - (erf(|x|)-c)| <= 2**-59.06
 *         Remark: here we use the taylor series expansion at x=1.
 *              erf(1+s) = erf(1) + s*Poly(s)
 *                       = 0.845.. + P1(s)/Q1(s)
 *         That is, we use rational approximation to approximate
 *                       erf(1+s) - (c = (single)0.84506291151)
 *         Note that |P1/Q1|< 0.078 for x in [0.84375,1.25]
 *         where
 *              P1(s) = degree 6 poly in s
 *              Q1(s) = degree 6 poly in s
 *
 *      3. For x in [1.25,1/0.35(~2.857143)],
 *              erfc(x) = (1/x)*exp(-x*x-0.5625+R1/S1)
 *              erf(x)  = 1 - erfc(x)
 *         where
 *              R1(z) = degree 7 poly in z, (z=1/x^2)
 *              S1(z) = degree 8 poly in z
 *
 *      4. For x in [1/0.35,28]
 *              erfc(x) = (1/x)*exp(-x*x-0.5625+R2/S2) if x > 0
 *                      = 2.0 - (1/x)*exp(-x*x-0.5625+R2/S2) if -6<x<0
 *                      = 2.0 - tiny            (if x <= -6)
 *              erf(x)  = sign(x)*(1.0 - erfc(x)) if x < 6, else
 *              erf(x)  = sign(x)*(1.0 - tiny)
 *         where
 *              R2(z) = degree 6 poly in z, (z=1/x^2)
 *              S2(z) = degree 7 poly in z
 *
 *      Note1:
 *         To compute exp(-x*x-0.5625+R/S), let s be a single
 *         precision number and s := x; then
 *              -x*x = -s*s + (s-x)*(s+x)
 *              exp(-x*x-0.5626+R/S) =
 *                      exp(-s*s-0.5625)*exp((s-x)*(s+x)+R/S);
 *      Note2:
 *         Here 4 and 5 make use of the asymptotic series
 *                            exp(-x*x)
 *              erfc(x) ~ ---------- * ( 1 + Poly(1/x^2) )
 *                            x*sqrt(pi)
 *         We use rational approximation to approximate
 *              g(s)=f(1/x^2) = log(erfc(x)*x) - x*x + 0.5625
 *         Here is the error bound for R1/S1 and R2/S2
 *              |R1/S1 - f(x)|  < 2**(-62.57)
 *              |R2/S2 - f(x)|  < 2**(-61.52)
 *
 *      5. For inf > x >= 28
 *              erf(x)  = sign(x) *(1 - tiny)  (raise inexact)
 *              erfc(x) = tiny*tiny (raise underflow) if x > 0
 *                      = 2 - tiny if x<0
 *
 *      7. Special case:
 *              erf(0)  = 0, erf(inf)  = 1, erf(-inf) = -1,
```

```
 *              erfc(0) = 1, erfc(inf) = 0, erfc(-inf) = 2,
 *      erfc/erf(NaN) is NaN
 */
/* INDENT ON */

#include "libm_macros.h"
#include <math.h>

static const double xxx[] = {
/* tiny */
        1e-300,
/* tiny */      1e-300,
/* half */ 5.00000000000000000000e-01,  /* 3FE00000, 00000000 */
/* one */ 1.00000000000000000000e+00,   /* 3FF00000, 00000000 */
/* two */ 2.00000000000000000000e+00,   /* 40000000, 00000000 */
/* erx */ 8.45062911510467529297e-01,   /* 3FEB0AC1, 60000000 */

#endif /* ! codereview */
/*
 * Coefficients for approximation to  erf on [0,0.84375]
 */
/* efx */ 1.28379167095512586316e-01,   /* 3FC06EBA, 8214DB69 */
/* efx8 */ 1.02703333676410069053e+00,  /* 3FF06EBA, 8214DB69 */
/* pp0 */ 1.28379167095512558561e-01,   /* 3FC06EBA, 8214DB68 */
/* pp1 */ -3.25042107247001499370e-01,  /* BFD4CD7D, 691CB913 */
/* pp2 */ -2.84817495755985104766e-02,  /* BF9D2A51, DBD7194F */
/* pp3 */ -5.77027029648944159157e-03,  /* BF77A291, 236668E4 */
/* pp4 */ -2.37630166566501626084e-05,  /* BEF8EAD6, 120016AC */
/* qq1 */ 3.97917223959155352819e-01,   /* 3FD97779, CDDADC09 */
/* qq2 */ 6.50222499887672944485e-02,   /* 3FB0A54C, 5536CEBA */
/* qq3 */ 5.08130628187576562776e-03,   /* 3F74D022, C4D36B0F */
/* qq4 */ 1.32494738004321644526e-04,   /* 3F215DC9, 221C1A10 */
/* qq5 */ -3.96022827877536812320e-06,  /* BED09C43, 42A26120 */

#endif /* ! codereview */
/*
 * Coefficients for approximation to  erf  in [0.84375,1.25]
 */
/* pa0 */ -2.36211856075265944077e-03,  /* BF6359B8, BEF77538 */
/* pa1 */ 4.14856118683748331666e-01,   /* 3FDA8D00, AD92B34D */
/* pa2 */ -3.72207876035701323847e-01,  /* BFD7D240, FBB8C3F1 */
/* pa3 */ 3.18346619901161753674e-01,   /* 3FD45FCA, 805120E4 */
/* pa4 */ -1.10894694282396677476e-01,  /* BFBC6398, 3D3E28EC */
/* pa5 */ 3.54783043256182359371e-02,   /* 3FA22A36, 599795EB */
/* pa6 */ -2.16637559486879084300e-03,  /* BF61BF38, 0A96073F */
/* qa1 */ 1.06420880400844228286e-01,   /* 3FBB3E66, 18EEE323 */
/* qa2 */ 5.40397917702171048937e-01,   /* 3FE14AF0, 92EB6F33 */
/* qa3 */ 7.18286544141962662868e-02,   /* 3FB2635C, D99FE9A7 */
/* qa4 */ 1.26171219808761642112e-01,   /* 3FC02660, E763351F */
/* qa5 */ 1.36370839120290507362e-02,   /* 3F8BEDC2, 6B51DD1C */
/* qa6 */ 1.19844998467991074170e-02,   /* 3F888B54, 5735151D */

#endif /* ! codereview */
/*
 * Coefficients for approximation to  erfc in [1.25,1/0.35]
 */
/* ra0 */ -9.86494403484714822705e-03,  /* BF843412, 600D6435 */
/* ra1 */ -6.93858572707181764372e-01,  /* BFE63416, E4BA7360 */
/* ra2 */ -1.05586262253232909814e+01,  /* C0251E04, 41B0E726 */
/* ra3 */ -6.23753324503260060396e+01,  /* C04F300A, E4CBA38D */
/* ra4 */ -1.62396669462573470355e+02,  /* C0644CB1, 84282266 */
/* ra5 */ -1.84605092906711035994e+02,  /* C067135C, EBCCABB2 */
/* ra6 */ -8.12874355063065934246e+01,  /* C0545265, 57E4D2F2 */
/* ra7 */ -9.81432934416914548592e+00,  /* C023A0EF, C69AC25C */
/* sa1 */ 1.96512716674392571292e+01,   /* 4033A6B9, BD707687 */
/* sa2 */ 1.37657754143519042600e+02,   /* 4061350C, 526AE721 */
```



```
/* sa3 */ 4.34565877475229228821e+02,   /* 407B290D, D58A1A71 */
/* sa4 */ 6.45387271733267880336e+02,   /* 40842B19, 21EC2868 */
/* sa5 */ 4.29008140027567833386e+02,   /* 407AD021, 57700314 */
/* sa6 */ 1.08635005541779435134e+02,   /* 405B28A3, EE48AE2C */
/* sa7 */ 6.57024977031928170135e+00,   /* 401A47EF, 8E484A93 */
/* sa8 */ -6.04244152148580987438e-02,  /* BFAEEFF2, EE749A62 */

#endif /* ! codereview */
/*
 * Coefficients for approximation to  erfc in [1/.35,28]
 */
/* rb0 */ -9.86494292470009928597e-03,  /* BF843412, 39E86F4A */
/* rb1 */ -7.99283237680523006574e-01,  /* BFE993BA, 70C285DE */
/* rb2 */ -1.77579549177547519889e+01,  /* C031C209, 555F995A */
/* rb3 */ -1.60636384855821916062e+02,  /* C064145D, 43C5ED98 */
/* rb4 */ -6.37566443368389627722e+02,  /* C083EC88, 1375F228 */
/* rb5 */ -1.02509513161107724954e+03,  /* C0900461, 6A2E5992 */
/* rb6 */ -4.83519191608651397019e+02,  /* C07E384E, 9BDC383F */
/* sb1 */ 3.03380607434824582924e+01,   /* 403E568B, 261D5190 */
/* sb2 */ 3.25792512996573918826e+02,   /* 40745CAE, 221B9F0A */
/* sb3 */ 1.53672958608443695994e+03,   /* 409802EB, 189D5118 */
/* sb4 */ 3.19985821950859553908e+03,   /* 40A8FFB7, 688C246A */
/* sb5 */ 2.55305040643316442583e+03,   /* 40A3F219, CEDF3BE6 */
/* sb6 */ 4.74528541206955367215e+02,   /* 407DA874, E79FE763 */
/* sb7 */ -2.24409524465858183362e+01   /* C03670E2, 42712D62 */
};

#define tiny            xxx[0]
#define half            xxx[1]
#define one             xxx[2]
#define two             xxx[3]
#define erx             xxx[4]

#endif /* ! codereview */
/*
 * Coefficients for approximation to  erf on [0,0.84375]
 */
#define efx             xxx[5]
#define efx8            xxx[6]
#define pp0             xxx[7]
#define pp1             xxx[8]
#define pp2             xxx[9]
#define pp3             xxx[10]
#define pp4             xxx[11]
#define qq1             xxx[12]
#define qq2             xxx[13]
#define qq3             xxx[14]
#define qq4             xxx[15]
#define qq5             xxx[16]

#endif /* ! codereview */
/*
 * Coefficients for approximation to  erf  in [0.84375,1.25]
 */
#define pa0             xxx[17]
#define pa1             xxx[18]
#define pa2             xxx[19]
#define pa3             xxx[20]
#define pa4             xxx[21]
#define pa5             xxx[22]
#define pa6             xxx[23]
#define qa1             xxx[24]
#define qa2             xxx[25]
#define qa3             xxx[26]
#define qa4             xxx[27]
#define qa5             xxx[28]
```

```
#define qa6             xxx[29]

#endif /* ! codereview */
/*
 * Coefficients for approximation to  erfc in [1.25,1/0.35]
 */
#define ra0             xxx[30]
#define ra1             xxx[31]
#define ra2             xxx[32]
#define ra3             xxx[33]
#define ra4             xxx[34]
#define ra5             xxx[35]
#define ra6             xxx[36]
#define ra7             xxx[37]
#define sa1             xxx[38]
#define sa2             xxx[39]
#define sa3             xxx[40]
#define sa4             xxx[41]
#define sa5             xxx[42]
#define sa6             xxx[43]
#define sa7             xxx[44]
#define sa8             xxx[45]

#endif /* ! codereview */
/*
 * Coefficients for approximation to  erfc in [1/.35,28]
 */
#define rb0             xxx[46]
#define rb1             xxx[47]
#define rb2             xxx[48]
#define rb3             xxx[49]
#define rb4             xxx[50]
#define rb5             xxx[51]
#define rb6             xxx[52]
#define sb1             xxx[53]
#define sb2             xxx[54]
#define sb3             xxx[55]
#define sb4             xxx[56]
#define sb5             xxx[57]
#define sb6             xxx[58]
#define sb7             xxx[59]

double
erf(double x)
{
erf(double x) {
        int hx, ix, i;
        double R, S, P, Q, s, y, z, r;

        hx = ((int *)&x)[HIWORD];
        hx = ((int *) &x)[HIWORD];
        ix = hx & 0x7fffffff;

#endif /* ! codereview */
        if (ix >= 0x7ff00000) {         /* erf(nan)=nan */
#if defined(FPADD_TRAPS_INCOMPLETE_ON_NAN)
                if (ix >= 0x7ff80000)   /* assumes sparc-like QNaN */
                        return (x);
#endif
                i = ((unsigned)hx >> 31) << 1;
                return ((double)(1 - i) + one / x);     /* erf(+-inf)=+-1 */
                i = ((unsigned) hx >> 31) << 1;
                return ((double) (1 - i) + one / x);    /* erf(+-inf)=+-1 */
        }

        if (ix < 0x3feb0000) {                          /* |x|<0.84375 */
```



```
                if (ix < 0x3e300000) {                  /* |x|<2**-28 */
                        if (ix < 0x00800000)            /* avoid underflow */
                                return (0.125 * (8.0 * x + efx8 * x));

#endif /* ! codereview */
                        return (x + efx * x);
                }

#endif /* ! codereview */
                z = x * x;
                r = pp0 + z * (pp1 + z * (pp2 + z * (pp3 + z * pp4)));
                s = one + z * (qq1 + z * (qq2 + z * (qq3 + z * (qq4 + z *
                    qq5))));
                s = one +
                        z *(qq1 + z * (qq2 + z * (qq3 + z * (qq4 + z * qq5))));
                y = r / s;
                return (x + x * y);
        }

#endif /* ! codereview */
        if (ix < 0x3ff40000) {          /* 0.84375 <= |x| < 1.25 */
                s = fabs(x) - one;
                P = pa0 + s * (pa1 + s * (pa2 + s * (pa3 + s * (pa4 + s * (pa5 +
                    s * pa6)))));
                Q = one + s * (qa1 + s * (qa2 + s * (qa3 + s * (qa4 + s * (qa5 +
                    s * qa6)))));

                P = pa0 + s * (pa1 + s * (pa2 + s * (pa3 + s * (pa4 +
                        s * (pa5 + s * pa6)))));
                Q = one + s * (qa1 + s * (qa2 + s * (qa3 + s * (qa4 +
                        s * (qa5 + s * qa6)))));
                if (hx >= 0)
                        return (erx + P / Q);
                else
                        return (-erx - P / Q);
        }

#endif /* ! codereview */
        if (ix >= 0x40180000) {         /* inf > |x| >= 6 */
                if (hx >= 0)
                        return (one - tiny);
                else
                        return (tiny - one);
        }

#endif /* ! codereview */
        x = fabs(x);
        s = one / (x * x);

#endif /* ! codereview */
        if (ix < 0x4006DB6E) {          /* |x| < 1/0.35 */
                R = ra0 + s * (ra1 + s * (ra2 + s * (ra3 + s * (ra4 + s * (ra5 +
                    s * (ra6 + s * ra7))))));
                S = one + s * (sa1 + s * (sa2 + s * (sa3 + s * (sa4 + s * (sa5 +
                    s * (sa6 + s * (sa7 + s * sa8)))))));
                R = ra0 + s * (ra1 + s * (ra2 + s * (ra3 + s * (ra4 +
                        s * (ra5 + s * (ra6 + s * ra7))))));
                S = one + s * (sa1 + s * (sa2 + s * (sa3 + s * (sa4 +
                        s * (sa5 + s * (sa6 + s * (sa7 + s * sa8)))))));
        } else {                        /* |x| >= 1/0.35 */
                R = rb0 + s * (rb1 + s * (rb2 + s * (rb3 + s * (rb4 + s * (rb5 +
                    s * rb6)))));
                S = one + s * (sb1 + s * (sb2 + s * (sb3 + s * (sb4 + s * (sb5 +
                    s * (sb6 + s * sb7))))));
                R = rb0 + s * (rb1 + s * (rb2 + s * (rb3 + s * (rb4 +
                        s * (rb5 + s * rb6)))));
```

```
166                 S = one + s * (sb1 + s * (sb2 + s * (sb3 + s * (sb4 +
167                         s * (sb5 + s * (sb6 + s * sb7))))));
373         }

375 #endif /* ! codereview */
376         z = x;
377         ((int *)&z)[LOWORD] = 0;
169         ((int *) &z)[LOWORD] = 0;
378         r = exp(-z * z - 0.5625) * exp((z - x) * (z + x) + R / S);

380 #endif /* ! codereview */
381         if (hx >= 0)
382                 return (one - r / x);
383         else
384                 return (r / x - one);
385 }

387 double
388 erfc(double x)
389 {
171 erfc(double x) {
390         int hx, ix;
391         double R, S, P, Q, s, y, z, r;

393         hx = ((int *)&x)[HIWORD];
175         hx = ((int *) &x)[HIWORD];
394         ix = hx & 0x7fffffff;

396 #endif /* ! codereview */
397         if (ix >= 0x7ff00000) {         /* erfc(nan)=nan */
398 #if defined(FPADD_TRAPS_INCOMPLETE_ON_NAN)
399                 if (ix >= 0x7ff80000)   /* assumes sparc-like QNaN */
400                         return (x);
401 #endif
402                 /* erfc(+-inf)=0,2 */
403                 return ((double)(((unsigned)hx >> 31) << 1) + one / x);
177                 return ((double) (((unsigned) hx >> 31) << 1) + one / x);
404         }

406         if (ix < 0x3feb0000) {          /* |x| < 0.84375 */
407                 if (ix < 0x3c700000)    /* |x| < 2**-56 */
408                         return (one - x);

410 #endif /* ! codereview */
411                 z = x * x;
412                 r = pp0 + z * (pp1 + z * (pp2 + z * (pp3 + z * pp4)));
413                 s = one + z * (qq1 + z * (qq2 + z * (qq3 + z * (qq4 + z *
414                     qq5))));
183                 s = one +
184                         z * (qq1 + z * (qq2 + z * (qq3 + z * (qq4 + z * qq5))));
415                 y = r / s;

417 #endif /* ! codereview */
418                 if (hx < 0x3fd00000) {  /* x < 1/4 */
419                         return (one - (x + x * y));
420                 } else {
421                         r = x * y;
422                         r += (x - half);
423                         return (half - r);
424                 }
425         }

427 #endif /* ! codereview */
428         if (ix < 0x3ff40000) {          /* 0.84375 <= |x| < 1.25 */
429                 s = fabs(x) - one;
430                 P = pa0 + s * (pa1 + s * (pa2 + s * (pa3 + s * (pa4 + s * (pa5 +
```



```
431                     s * pa6)))));
432                 Q = one + s * (qa1 + s * (qa2 + s * (qa3 + s * (qa4 + s * (qa5 +
433                     s * qa6)))));

186                 P = pa0 + s * (pa1 + s * (pa2 + s * (pa3 + s * (pa4 +
187                         s * (pa5 + s * pa6))))));
188                 Q = one + s * (qa1 + s * (qa2 + s * (qa3 + s * (qa4 +
189                         s * (qa5 + s * qa6)))));
435                 if (hx >= 0) {
436                         z = one - erx;
437                         return (z - P / Q);
438                 } else {
439                         z = erx + P / Q;
440                         return (one + z);
441                 }
442         }

444 #endif /* ! codereview */
445         if (ix < 0x403c0000) {          /* |x|<28 */
446                 x = fabs(x);
447                 s = one / (x * x);

449 #endif /* ! codereview */
450                 if (ix < 0x4006DB6D) {  /* |x| < 1/.35 ~ 2.857143 */
451                         R = ra0 + s * (ra1 + s * (ra2 + s * (ra3 + s * (ra4 +
452                             s * (ra5 + s * (ra6 + s * ra7)))))));
453                         S = one + s * (sa1 + s * (sa2 + s * (sa3 + s * (sa4 +
454                             s * (sa5 + s * (sa6 + s * (sa7 + s * sa8)))))));
455                 } else {
456                         /* |x| >= 1/.35 ~ 2.857143 */
457                         if (hx < 0 && ix >= 0x40180000)
458                                 return (two - tiny);    /* x < -6 */

460                         R = rb0 + s * (rb1 + s * (rb2 + s * (rb3 + s * (rb4 +
461                             s * (rb5 + s * rb6)))));
462                         S = one + s * (sb1 + s * (sb2 + s * (sb3 + s * (sb4 +
463                             s * (sb5 + s * (sb6 + s * sb7))))));
464                 }

466 #endif /* ! codereview */
467                 z = x;
468                 ((int *)&z)[LOWORD] = 0;
198                 ((int *) &z)[LOWORD] = 0;
469                 r = exp(-z * z - 0.5625) * exp((z - x) * (z + x) + R / S);

471 #endif /* ! codereview */
472                 if (hx > 0)
473                         return (r / x);
474                 else
475                         return (two - r / x);
476         } else {
477                 if (hx > 0)
478                         return (tiny * tiny);
479                 else
480                         return (two - tiny);
481         }
482 }
```

```
**********************************************************
   14893 Thu Jun  6 20:37:44 2019
new/usr/src/lib/libm/common/C/exp.c
11210 libm should be cstyle(10NBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

#endif /* ! codereview */
/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __exp = exp

/*
 * exp(x)
 * Hybrid algorithm of Peter Tang's Table driven method (for large
 * arguments) and an accurate table (for small arguments).
 * Written by K.C. Ng, November 1988.
 * Method (large arguments):
 *      1. Argument Reduction: given the input x, find r and integer k
 *         and j such that
 *                   x = (k+j/32)*(ln2) + r,  |r| <= (1/64)*ln2
 *
 *      2. exp(x) = 2^k * (2^(j/32) + 2^(j/32)*expm1(r))
 *         a. expm1(r) is approximated by a polynomial:
 *            expm1(r) ~ r + t1*r^2 + t2*r^3 + ... + t5*r^6
 *            Here t1 = 1/2 exactly.
 *         b. 2^(j/32) is represented to twice double precision
 *            as TBL[2j]+TBL[2j+1].
 *
 * Note: If divide were fast enough, we could use another approximation
 *       in 2.a:
 *            expm1(r) ~ (2r)/(2-R), R = r - r^2*(t1 + t2*r^2)
 *            (for the same t1 and t2 as above)
 *
 * Special cases:
 *      exp(INF) is INF, exp(NaN) is NaN;
 *      exp(-INF)=  0;
 *      for finite argument, only exp(0)=1 is exact.
 *
```



```
 * Accuracy:
 *      According to an error analysis, the error is always less than
 *      an ulp (unit in the last place).  The largest errors observed
 *      are less than 0.55 ulp for normal results and less than 0.75 ulp
 *      for subnormal results.
 *
 * Misc. info.
 *      For IEEE double
 *              if x >  7.09782712893383973096e+02 then exp(x) overflow
 *              if x < -7.45133219101941108420e+02 then exp(x) underflow
 */

#include "libm.h"

static const double TBL[] = {
        1.00000000000000000000e+00, 0.00000000000000000000e+00,
        1.02189714865411662714e+00, 5.10922502897344389359e-17,
        1.04427378242741375480e+00, 8.55188970553796365958e-17,
        1.06714040067682369717e+00, -7.89985396684158212226e-17,
        1.09050773266525768967e+00, -3.04678207981247114697e-17,
        1.11438674259589243221e+00, 1.04102784568455709549e-16,
        1.13878863475669156458e+00, 8.91281267602540777782e-17,
        1.16372485877757747552e+00, 3.82920483692409349872e-17,
        1.18920711500272102690e+00, 3.98201523146564611098e-17,
        1.21524735998046895524e+00, -7.71263069268148813091e-17,
        1.24185781207348400201e+00, 4.65802759183693679123e-17,
        1.26905095719173321989e+00, 2.66793213134218609523e-18,
        1.29683955465100964055e+00, 2.53825027948883149593e-17,
        1.32523664315974132322e+00, -2.85873121003886075697e-17,
        1.35425554693689265129e+00, 7.70094837980298946162e-17,
        1.38390988196383202258e+00, -6.77051165879478628716e-17,
        1.41421356237309514547e+00, -9.66729331345291345105e-17,
        1.44518080697704665027e+00, -3.02375813499398731940e-17,
        1.47682614593949934623e+00, -3.48399455689279579579e-17,
        1.50916442759342284141e+00, -1.01645532775429503911e-16,
        1.54221082540794074411e+00, 7.94983480969762085616e-17,
        1.57598084510788649659e+00, -1.01369164712783039808e-17,
        1.61049033194925428347e+00, 2.47071925697978878522e-17,
        1.64575547815396494578e+00, -1.01256799136747726038e-16,
        1.68179283050742900407e+00, 8.19901002058149652013e-17,
        1.71861929812247793414e+00, -1.85138041826311098821e-17,
        1.75625216037329945351e+00, 2.96014069544887330703e-17,
        1.79470907500310716820e+00, 1.82274584279120867698e-17,
        1.83400808640934243066e+00, 3.28310722424562658722e-17,
        1.87416763411029996256e+00, -6.12276341300414256164e-17,
        1.91520656139714740007e+00, -1.06199460561959626376e-16,
        1.95714412417540017941e+00, 8.96076779103666776760e-17,
};

/*
 * For i = 0, ..., 66,
 *   TBL2[2*i] is a double precision number near (i+1)*2^-6, and
 *   TBL2[2*i+1] = exp(TBL2[2*i]) to within a relative error less
 *   than 2^-60.
 *
 * For i = 67, ..., 133,
 *   TBL2[2*i] is a double precision number near -(i+1)*2^-6, and
 *   TBL2[2*i+1] = exp(TBL2[2*i]) to within a relative error less
 *   than 2^-60.
 */
static const double TBL2[] = {
        1.56249999999984491572e-02, 1.01574770858668417262e+00,
        3.12499999999998716305e-02, 1.03174340749910253834e+00,
        4.68750000000011102230e-02, 1.04799100201663386578e+00,
        6.24999999999990632493e-02, 1.06449445891785843266e+00,
        7.81249999999999444888e-02, 1.08125780744903954300e+00,
```

```
        9.37500000000013322676e-02, 1.09828514030782731226e+00,
        1.09375000000001346145e-01, 1.11558061464248226002e+00,
        1.24999999999999417133e-01, 1.13314845306682565607e+00,
        1.40624999999995337063e-01, 1.15099294469117108264e+00,
        1.56249999999996141975e-01, 1.16911844616949989195e+00,
        1.71874999999992894573e-01, 1.18752938276309216725e+00,
        1.87500000000000888178e-01, 1.20623024942098178158e+00,
        2.03124999999361649516e-01, 1.22522561187652545556e+00,
        2.18750000000000416334e-01, 1.24452010776609567344e+00,
        2.34375000000003524958e-01, 1.26411844775347081971e+00,
        2.50000000000006328271e-01, 1.28402541668774961003e+00,
        2.65624999999982791543e-01, 1.30424587476761533189e+00,
        2.81249999999993727240e-01, 1.32478475872885725906e+00,
        2.96875000000003275158e-01, 1.34564708304941493822e+00,
        3.12500000000002886580e-01, 1.36683794117380030819e+00,
        3.28124999999993394173e-01, 1.38836250675661765364e+00,
        3.43749999999998612221e-01, 1.41022603492570874906e+00,
        3.59374999999992450483e-01, 1.43243386356506730017e+00,
        3.74999999999991395772e-01, 1.45499141461818881638e+00,
        3.90624999999997613020e-01, 1.47790419541173490003e+00,
        4.06249999999991895372e-01, 1.50117780000011058483e+00,
        4.21874999999996613820e-01, 1.52481791053132154090e+00,
        4.37500000000004607426e-01, 1.54883029863414023453e+00,
        4.53125000000004274359e-01, 1.57322082682725961078e+00,
        4.68750000000008326673e-01, 1.59799544995064657371e+00,
        4.84374999999985456078e-01, 1.62316021661928200359e+00,
        4.99999999999997335465e-01, 1.64872127070012375327e+00,
        5.15625000000000222045e-01, 1.67468485281178436352e+00,
        5.31250000000003441691e-01, 1.70105730184840653330e+00,
        5.46874999999999111822e-01, 1.72784505652716169344e+00,
        5.62499999999999333866e-01, 1.75505465696029738787e+00,
        5.78124999999993338662e-01, 1.78269274625180318417e+00,
        5.93749999999999666933e-01, 1.81076607211938656050e+00,
        6.09375000000003441691e-01, 1.83928148854178719063e+00,
        6.24999999999995559108e-01, 1.86824595743221411048e+00,
        6.40625000000009103829e-01, 1.89766655033813602671e+00,
        6.56249999999993782751e-01, 1.92755045016753268072e+00,
        6.71875000000002109424e-01, 1.95790495294292221651e+00,
        6.87499999999992450483e-01, 1.98873746958227681780e+00,
        7.03125000000004996004e-01, 2.02005552770870666635e+00,
        7.18750000000007105427e-01, 2.05186677348799140219e+00,
        7.34375000000008770762e-01, 2.08417897349558689513e+00,
        7.49999999999983901766e-01, 2.11700016612640589939e+00,
        7.65624999999997002398e-01, 2.15033791595229351046e+00,
        7.81250000000005884182e-01, 2.18420081081563077774e+00,
        7.96874999999991451283e-01, 2.21859696867912603579e+00,
        8.12500000000000000000e-01, 2.25353478721320854561e+00,
        8.28125000000008215650e-01, 2.28902279633221983346e+00,
        8.43749999999997890576e-01, 2.32506966027711614586e+00,
        8.59374999999999444888e-01, 2.36168417973090827289e+00,
        8.75000000000003219647e-01, 2.39887529396710563745e+00,
        8.90625000000013433699e-01, 2.43665208303232461162e+00,
        9.06249999999980571097e-01, 2.47502376996297712708e+00,
        9.21874999999984456878e-01, 2.51399972303748420188e+00,
        9.37500000000001887379e-01, 2.55358945806293169412e+00,
        9.53125000000003330669e-01, 2.59380264069854327147e+00,
        9.68749999999989119814e-01, 2.63464908881560244680e+00,
        9.84374999999997890576e-01, 2.67613877489447116176e+00,
        1.00000000000000154632e+00, 2.71828182845907662113e+00,
        1.01562499999999333866e+00, 2.76108853855008318234e+00,
        1.03124999999995980993e+00, 2.80456935623711389738e+00,
        1.04687499999999933387e+00, 2.84873489717039740654e+00,
       -1.56249999999999514277e-02, 9.84496437005408453480e-01,
       -3.12499999999955972718e-02, 9.69233234476348348707e-01,
       -4.68749999999993824384e-02, 9.54206665969188905230e-01,
       -6.24999999999976130205e-02, 9.39413062813478028090e-01,
```



```
       -7.81249999999989314103e-02, 9.24848813216205822840e-01,
       -9.37499999999995975442e-02, 9.10510361380034494161e-01,
       -1.09374999999998584466e-01, 8.96394206635151680196e-01,
       -1.24999999999998556710e-01, 8.82496902584596676355e-01,
       -1.40624999999999361622e-01, 8.68815056262843721235e-01,
       -1.56249999999999111822e-01, 8.55345327307423297647e-01,
       -1.71874999999924144012e-01, 8.42084427143446223596e-01,
       -1.87499999999996752598e-01, 8.29029118180403035154e-01,
       -2.03124999999988037347e-01, 8.16176213022349550386e-01,
       -2.18749999999995947686e-01, 8.03522573689063990265e-01,
       -2.34374999999996419531e-01, 7.91065110850298847112e-01,
       -2.49999999999996280753e-01, 7.78800783071407765057e-01,
       -2.65624999999999888978e-01, 7.66726596070820165529e-01,
       -2.81249999999989397370e-01, 7.54839601989015340777e-01,
       -2.96874999999996114219e-01, 7.43136898668761203268e-01,
       -3.12499999999999555911e-01, 7.31615628946642115871e-01,
       -3.28124999999993782751e-01, 7.20272979955444259126e-01,
       -3.43749999999997946087e-01, 7.09106182437399867879e-01,
       -3.59374999999994337863e-01, 6.98112510068129790023e-01,
       -3.74999999999994615418e-01, 6.87289278790975899369e-01,
       -3.90624999999999000799e-01, 6.76633846161729612945e-01,
       -4.06249999999947264406e-01, 6.66143610703522903727e-01,
       -4.21874999999988453681e-01, 6.55816011271509125002e-01,
       -4.37499999999999111822e-01, 6.45648526427892610613e-01,
       -4.53124999999999278355e-01, 6.35638673826052436056e-01,
       -4.68749999999999278355e-01, 6.25784009604591573428e-01,
       -4.84374999999992894573e-01, 6.16082127790682609891e-01,
       -4.99999999999998168132e-01, 6.06530659712634534486e-01,
       -5.15625000000000000000e-01, 5.97127273421627413619e-01,
       -5.31249999999989785948e-01, 5.87869673122352498496e-01,
       -5.46874999999972688514e-01, 5.78755598612500032907e-01,
       -5.62500000000000000000e-01, 5.69782824730923009859e-01,
       -5.78124999999992339461e-01, 5.60949160814475100700e-01,
       -5.93749999999948707696e-01, 5.52252450163048691500e-01,
       -6.09374999999552580121e-01, 5.43690569513243682209e-01,
       -6.24999999999984789945e-01, 5.35261428518998383375e-01,
       -6.40624999999983457677e-01, 5.26962969243379708573e-01,
       -6.56249999999998334665e-01, 5.18793165653890220312e-01,
       -6.71874999999943378626e-01, 5.10750023129039609771e-01,
       -6.87499999999997002398e-01, 5.02831577970942467104e-01,
       -7.03124999999991118216e-01, 4.95035896926202978463e-01,
       -7.18749999999991340260e-01, 4.87361076713623331269e-01,
       -7.34374999999985678123e-01, 4.79805243559684402310e-01,
       -7.49999999999997335465e-01, 4.72366552741015965911e-01,
       -7.65624999999993782751e-01, 4.65043188134059204408e-01,
       -7.81249999999863220523e-01, 4.57833361771676883301e-01,
       -7.96874999999998112621e-01, 4.50735313406363247157e-01,
       -8.12499999999990119015e-01, 4.43747310081084256339e-01,
       -8.28124999999996003197e-01, 4.36867645705559026759e-01,
       -8.43749999999988120614e-01, 4.30094640640067360504e-01,
       -8.59374999999994115818e-01, 4.23426641285265303871e-01,
       -8.74999999999977129406e-01, 4.16862019678517936594e-01,
       -8.90624999999983346655e-01, 4.10399173096376801428e-01,
       -9.06249999999991784350e-01, 4.04036523663345414903e-01,
       -9.21874999999994004796e-01, 3.97772517966614058693e-01,
       -9.37499999999994337863e-01, 3.91605626676801210628e-01,
       -9.53124999999999444888e-01, 3.85534344174578935682e-01,
       -9.68749999999986677324e-01, 3.79557188183094640355e-01,
       -9.84374999999992339461e-01, 3.73672699406045860648e-01,
       -9.99999999999995892175e-01, 3.67879441171443832825e-01,
       -1.01562499999999315658e+00, 3.62175999080846300338e-01,
       -1.03124999999991096011e+00, 3.56560980663978732697e-01,
       -1.04687499999999067413e+00, 3.51033015038813400732e-01,
};

static const double C[] = {
```

```
        0.5,
        4.6166241308446828384 1e+01,     /* 0x40471547, 0x652b82fe */
        2.1660849386535119265 3e-02,     /* 0x3f962e42, 0xfee00000 */
        5.9631716539705865625 7e-12,     /* 0x3d9a39ef, 0x35793c76 */
        1.6666666666526086527e-1,        /* 3fc5555555548f7c */
        4.1666666666226079285e-2,        /* 3fa5555555545d4e */
        8.3333679843421958056e-3,        /* 3f811115b7aa905e */
        1.3888949086377719040e-3,        /* 3f56c1728d739765 */
        1.0,
        0.0,
        7.0978271289338397309 6e+02, /* 0x40862E42, 0xFEFA39EF */
        7.4513321910194110842 0e+02, /* 0x40874910, 0xD52D3051 */
        5.5511151231257827021 2e-17, /* 0x3c900000, 0x00000000 */
};

#define half                    C[0]
#define invln2_32               C[1]
#define ln2_32hi                C[2]
#define ln2_32lo                C[3]
#define t2                      C[4]
#define t3                      C[5]
#define t4                      C[6]
#define t5                      C[7]
#define one                     C[8]
#define zero                    C[9]
#define threshold1              C[10]
#define threshold2              C[11]
#define twom54                  C[12]

double
exp(double x)
{
exp(double x) {
        double y, z, t;
        int hx, ix, k, j, m;

        ix = ((int *)&x)[HIWORD];
        hx = ix & ~0x80000000;

        if (hx < 0x3ff0a2b2) {                  /* |x| < 3/2 ln 2 */
                if (hx < 0x3f862e42) {          /* |x| < 1/64 ln 2 */
                        if (hx < 0x3ed00000) {  /* |x| < 2^-18 */
                                volatile int dummy __unused;

                                dummy = (int)x; /* raise inexact if x != 0 */
#ifdef lint
                                dummy = dummy;
#endif

#endif /* ! codereview */
                                if (hx < 0x3e300000)
                                        return (one + x);

#endif /* ! codereview */
                                return (one + x * (one + half * x));
                        }

#endif /* ! codereview */
                        t = x * x;
                        y = x + (t * (half + x * t2) + (t * t) * (t3 + x * t4 +
                            t * t5));
                        y = x + (t * (half + x * t2) +
                            (t * t) * (t3 + x * t4 + t * t5));
                        return (one + y);
                }
```



```
                /* find the multiple of 2^-6 nearest x */
                k = hx >> 20;
                j = (0x00100000 | (hx & 0x000fffff)) >> (0x40c - k);
                j = (j - 1) & ~1;

#endif /* ! codereview */
                if (ix < 0)
                        j += 134;

#endif /* ! codereview */
                z = x - TBL2[j];
                t = z * z;
                y = z + (t * (half + z * t2) + (t * t) * (t3 + z * t4 + t *
                    t5));
                return (TBL2[j + 1] + TBL2[j + 1] * y);
                y = z + (t * (half + z * t2) +
                    (t * t) * (t3 + z * t4 + t * t5));
                return (TBL2[j+1] + TBL2[j+1] * y);
        }

        if (hx >= 0x40862e42) {         /* x is large, infinite, or nan */
                if (hx >= 0x7ff00000) {
                        if (ix == 0xfff00000 && ((int *)&x)[LOWORD] == 0)
                                return (zero);

#endif /* ! codereview */
                        return (x * x);
                }

#endif /* ! codereview */
                if (x > threshold1)
                        return (_SVID_libm_err(x, x, 6));

#endif /* ! codereview */
                if (-x > threshold2)
                        return (_SVID_libm_err(x, x, 7));
        }

        t = invln2_32 * x;

#endif /* ! codereview */
        if (ix < 0)
                t -= half;
        else
                t += half;

#endif /* ! codereview */
        k = (int)t;
        j = (k & 0x1f) << 1;
        m = k >> 5;
        z = (x - k * ln2_32hi) - k * ln2_32lo;

        /* z is now in primary range */
        t = z * z;
        y = z + (t * (half + z * t2) + (t * t) * (t3 + z * t4 + t * t5));
        y = TBL[j] + (TBL[j + 1] + TBL[j] * y);

        y = TBL[j] + (TBL[j+1] + TBL[j] * y);
        if (m < -1021) {
                ((int *)&y)[HIWORD] += (m + 54) << 20;
                return (twom54 * y);
        }

#endif /* ! codereview */
        ((int *)&y)[HIWORD] += m << 20;
        return (y);
```

```
}
```

```
**********************************************************
    2576 Thu Jun  6 20:37:45 2019
new/usr/src/lib/libm/common/C/exp10.c
11210 libm should be cstyle(10NBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

#endif /* ! codereview */
/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */


/* INDENT OFF */
/*
 * exp10(x)
 * Code by K.C. Ng for SUN 4.0 libm.
 * Method :
 *      n = nint(x*(log10/log2));
 *      exp10(x) = 10**x = exp(x*ln(10)) = exp(n*ln2+(x*ln10-n*ln2))
 *               = 2**n*exp(ln10*(x-n*log2/log10)))
 *      If x is an integer < 23 then use repeat multiplication. For
 *      10**22 is the largest representable integer.
 */
/* INDENT ON */

#include "libm.h"

static const double C[] = {
        3.3219280948736234787,          /* log(10)/log(2) */
        2.3025850929940456840,          /* log(10) */
        3.0102999565860955045E-1,       /* log(2)/log(10) high */
        5.3716447674669983622E-12,      /* log(2)/log(10) low */
        0.0,
        0.5,
        1.0,
        10.0,
        1.0e300,
        1.0e-300,
};
```



```
#define lg10            C[0]
#define ln10            C[1]
#define logt2hi         C[2]
#define logt2lo         C[3]
#define zero            C[4]
#define half            C[5]
#define one             C[6]
#define ten             C[7]
#define huge            C[8]
#define tiny            C[9]

double
exp10(double x)
{
exp10(double x) {
        double t, pt;
        int ix, hx, k;

        ix = ((int *)&x)[HIWORD];
        hx = ix & ~0x80000000;

        if (hx >= 0x4074a000) {         /* |x| >= 330 or x is nan */
                if (hx >= 0x7ff00000) { /* x is inf or nan */
                        if (ix == 0xfff00000 && ((int *)&x)[LOWORD] == 0)
                                return (zero);

#endif /* ! codereview */
                        return (x * x);
                }

                t = (ix < 0) ? tiny : huge;
                t = (ix < 0)? tiny : huge;
                return (t * t);
        }

        if (hx < 0x3c000000)
                return (one + x);

        k = (int)x;

#endif /* ! codereview */
        if (0 <= k && k < 23 && (double)k == x) {
                /* x is a small positive integer */
                t = one;
                pt = ten;

#endif /* ! codereview */
                if (k & 1)
                        t = ten;

#endif /* ! codereview */
                k >>= 1;

#endif /* ! codereview */
                while (k) {
                        pt *= pt;

#endif /* ! codereview */
                        if (k & 1)
                                t *= pt;

#endif /* ! codereview */
                        k >>= 1;
                }

#endif /* ! codereview */
```

```
                return (t);
        }

#endif /* ! codereview */
        t = x * lg10;
        k = (int)((ix < 0) ? t - half : t + half);
        k = (int)((ix < 0)? t - half : t + half);
        return (scalbn(exp(ln10 * ((x - k * logt2hi) - k * logt2lo)), k));
}
```

_____*unchanged_portion_omitted_*

```
**********************************************************
    2052 Thu Jun  6 20:37:45 2019
new/usr/src/lib/libm/common/C/exp2.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

#endif /* ! codereview */
/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __exp2 = exp2


/* INDENT OFF */
/*
 * exp2(x)
 * Code by K.C. Ng for SUN 4.0 libm.
 * Method :
 *      exp2(x) = 2**x = 2**((x-anint(x))+anint(x))
 *              = 2**anint(x)*2**(x-anint(x))
 *              = 2**anint(x)*exp((x-anint(x))*ln2)
 */
/* INDENT ON */

#include "libm.h"

static const double C[] = {
        0.0,
        1.0,
        0.5,
        6.93147180559945286227e-01,
        1.0e300,
        1.0e-300,
};

#define zero            C[0]
#define one             C[1]
#define half            C[2]
#define ln2             C[3]
```



```
#define huge            C[4]
#define tiny            C[5]

double
exp2(double x)
{
exp2(double x) {
        int ix, hx, k;
        double t;

        ix = ((int *)&x)[HIWORD];
        hx = ix & ~0x80000000;

        if (hx >= 0x4090e000) {         /* |x| >= 1080 or x is nan */
                if (hx >= 0x7ff00000) { /* x is inf or nan */
                        if (ix == 0xfff00000 && ((int *)&x)[LOWORD] == 0)
                                return (zero);

#endif /* ! codereview */
                        return (x * x);
                }

                t = (ix < 0) ? tiny : huge;
                t = (ix < 0)? tiny : huge;
                return (t * t);
        }

        if (hx < 0x3fe00000) {          /* |x| < 0.5 */
                if (hx < 0x3c000000)
                        return (one + x);

#endif /* ! codereview */
                return (exp(ln2 * x));
        }

        k = (int)x;

#endif /* ! codereview */
        if (x != (double)k)
                k = (int)((ix < 0) ? x - half : x + half);

                k = (int)((ix < 0)? x - half : x + half);
        return (scalbn(exp(ln2 * (x - (double)k)), k));
}
______unchanged_portion_omitted_
```

```
**********************************************************
    8510 Thu Jun  6 20:37:46 2019
new/usr/src/lib/libm/common/C/expm1.c
11210 libm should be cstyle(10NBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __expm1 = expm1


/* INDENT OFF */
/*
 * expm1(x)
 * Returns exp(x)-1, the exponential of x minus 1.
 *
 * Method
 *   1. Arugment reduction:
 *      Given x, find r and integer k such that
 *
 *               x = k*ln2 + r,  |r| <= 0.5*ln2 ~ 0.34658
 *
 *      Here a correction term c will be computed to compensate
 *      the error in r when rounded to a floating-point number.
 *
 *   2. Approximating expm1(r) by a special rational function on
 *      the interval [0,0.34658]:
 *      Since
 *          r*(exp(r)+1)/(exp(r)-1) = 2+ r^2/6 - r^4/360 + ...
 *      we define R1(r*r) by
 *          r*(exp(r)+1)/(exp(r)-1) = 2+ r^2/6 * R1(r*r)
 *      That is,
 *          R1(r**2) = 6/r *((exp(r)+1)/(exp(r)-1) - 2/r)
 *                   = 6/r * ( 1 + 2.0*(1/(exp(r)-1) - 1/r))
 *                   = 1 - r^2/60 + r^4/2520 - r^6/100800 + ...
 *      We use a special Reme algorithm on [0,0.347] to generate
 *      a polynomial of degree 5 in r*r to approximate R1. The
 *      maximum error of this polynomial approximation is bounded
```



```
 *      by 2**-61. In other words,
 *          R1(z) ~ 1.0 + Q1*z + Q2*z**2 + Q3*z**3 + Q4*z**4 + Q5*z**5
 *      where   Q1  =  -1.6666666666666567384E-2,
 *              Q2  =   3.9682539681370365873E-4,
 *              Q3  =  -9.9206344733435987357E-6,
 *              Q4  =   2.5051361420808517002E-7,
 *              Q5  =  -6.2843505682382617102E-9;
 *      (where z=r*r, and the values of Q1 to Q5 are listed below)
 *      with error bounded by
 *          |                                |     -61
 *          | 1.0+Q1*z+...+Q5*z   -  R1(z) | <= 2
 *          |                                |
 *
 *      expm1(r) = exp(r)-1 is then computed by the following
 *      specific way which minimize the accumulation rounding error:
 *                                2     3
 *                               r     r    [ 3 - (R1 + R1*r/2)  ]
 *            expm1(r) = r + --- + --- * [---------------------]
 *                               2     2    [ 6 - r*(3 - R1*r/2) ]
 *
 *      To compensate the error in the argument reduction, we use
 *              expm1(r+c) = expm1(r) + c + expm1(r)*c
 *                         ~ expm1(r) + c + r*c
 *      Thus c+r*c will be added in as the correction terms for
 *      expm1(r+c). Now rearrange the term to avoid optimization
 *      screw up:
 *                      (      2                                    2 )
 *                      ({  ( r    [ R1 -  (3 - R1*r/2) ]  )  }    r  )
 *       expm1(r+c)~r - ({r*(--- * [--------------------]-c)-c} - --- )
 *                      ({  ( 2    [ 6 - r*(3 - R1*r/2) ]  )  }    2  )
 *                      (                                              )
 *
 *                 = r - E
 *   3. Scale back to obtain expm1(x):
 *      From step 1, we have
 *         expm1(x) = either 2^k*[expm1(r)+1] - 1
 *                  = or     2^k*[expm1(r) + (1-2^-k)]
 *   4. Implementation notes:
 *      (A). To save one multiplication, we scale the coefficient Qi
 *           to Qi*2^i, and replace z by (x^2)/2.
 *      (B). To achieve maximum accuracy, we compute expm1(x) by
 *        (i)   if x < -56*ln2, return -1.0, (raise inexact if x != inf)
 *        (ii)  if k=0, return r-E
 *        (iii) if k=-1, return 0.5*(r-E)-0.5
 *        (iv)  if k=1 if r < -0.25, return 2*((r+0.5)- E)
 *                                   else         return  1.0+2.0*(r-E);
 *        (v)   if (k<-2||k>56) return 2^k(1-(E-r)) - 1 (or exp(x)-1)
 *        (vi)  if k <= 20, return 2^k((1-2^-k)-(E-r)), else
 *        (vii) return 2^k(1-((E+2^-k)-r))
 *
 * Special cases:
 *      expm1(INF) is INF, expm1(NaN) is NaN;
 *      expm1(-INF) is -1, and
 *      for finite argument, only expm1(0)=0 is exact.
 *
 * Accuracy:
 *      according to an error analysis, the error is always less than
 *      1 ulp (unit in the last place).
 *
 * Misc. info.
 *      For IEEE double
 *          if x >  7.09782712893383973096e+02 then expm1(x) overflow
 *
 * Constants:
 * The hexadecimal values are the intended ones for the following
 * constants. The decimal values may be used, provided that the
```

```
 * compiler will convert from decimal to binary accurately enough
 * to produce the hexadecimal values shown.
 */
/* INDENT ON */

#include "libm_macros.h"
#include <math.h>

static const double xxx[] = {
/* one */
        1.0,
/* one */                1.0,
/* huge */ 1.0e+300,
/* tiny */ 1.0e-300,
/* o_threshold */ 7.09782712893383973096e+02,   /* 40862E42 FEFA39EF */
/* ln2_hi */ 6.93147180369123816490e-01,        /* 3FE62E42 FEE00000 */
/* ln2_lo */ 1.90821492927058770002e-10,        /* 3DEA39EF 35793C76 */
/* invln2 */ 1.44269504088896338700e+00,        /* 3FF71547 652B82FE */

/*
 * scaled coefficients related to expm1
 * Q1
 */
        -3.33333333333331316428e-02,            /* BFA11111 111110F4 */
/* scaled coefficients related to expm1 */
/* Q1 */                -3.33333333333331316428e-02,    /* BFA11111 111110F4 */
/* Q2 */ 1.58730158725481460165e-03,            /* 3F5A01A0 19FE5585 */
/* Q3 */ -7.93650757867487942473e-05,           /* BF14CE19 9EAADBB7 */
/* Q4 */ 4.00821782732936239552e-06,            /* 3ED0CFCA 86E65239 */
/* Q5 */ -2.01099218183624371326e-07            /* BE8AFDB7 6E09C32D */
};

#endif /* ! codereview */
#define one                     xxx[0]
#define huge                    xxx[1]
#define tiny                    xxx[2]
#define o_threshold             xxx[3]
#define ln2_hi                  xxx[4]
#define ln2_lo                  xxx[5]
#define invln2                  xxx[6]
#define Q1                      xxx[7]
#define Q2                      xxx[8]
#define Q3                      xxx[9]
#define Q4                      xxx[10]
#define Q5                      xxx[11]

double
expm1(double x)
{
expm1(double x) {
        double y, hi, lo, c = 0.0L, t, e, hxs, hfx, r1;
        int k, xsb;
        unsigned hx;

        hx = ((unsigned *)&x)[HIWORD];  /* high word of x */
        hx = ((unsigned *) &x)[HIWORD];         /* high word of x */
        xsb = hx & 0x80000000;          /* sign bit of x */

#endif /* ! codereview */
        if (xsb == 0)
                y = x;
        else
                y = -x;                 /* y = |x| */

#endif /* ! codereview */
        hx &= 0x7fffffff;               /* high word of |x| */
```



```
        /*
         * filter out huge and non-finite argument
         * for example exp(38)-1 is approximately 3.1855932e+16
         */
        /* filter out huge and non-finite argument */
        /* for example exp(38)-1 is approximately 3.1855932e+16 */
        if (hx >= 0x4043687A) {
                /* if |x|>=56*ln2 (~38.8162...) */
                if (hx >= 0x40862E42) { /* if |x|>=709.78... -> inf */
                        if (hx >= 0x7ff00000) {
                                if (((hx & 0xfffff) | ((int *)&x)[LOWORD]) != 0)
                                if (((hx & 0xfffff) | ((int *) &x)[LOWORD])
                                        != 0)
                                        return (x * x); /* + -> * for Cheetah */
                                else
                                        /* exp(+-inf)={inf,-1} */
                                        return (xsb == 0 ? x : -1.0);
                        }

#endif /* ! codereview */
                        if (x > o_threshold)
                                return (huge * huge);   /* overflow */
                }

#endif /* ! codereview */
                if (xsb != 0) { /* x < -56*ln2, return -1.0 w/inexact */
                        if (x + tiny < 0.0)             /* raise inexact */
                                return (tiny - one);    /* return -1 */
                }
        }

        /* argument reduction */
        if (hx > 0x3fd62e42) {          /* if  |x| > 0.5 ln2 */
                if (hx < 0x3FF0A2B2) {  /* and |x| < 1.5 ln2 */
                        if (xsb == 0) { /* positive number */
                                hi = x - ln2_hi;
                                lo = ln2_lo;
                                k = 1;
                        } else {
                                /* negative number */
                                hi = x + ln2_hi;
                                lo = -ln2_lo;
                                k = -1;
                        }
                } else {
                        /* |x| > 1.5 ln2 */
                        k = (int)(invln2 * x + (xsb == 0 ? 0.5 : -0.5));
                        k = (int) (invln2 * x + (xsb == 0 ? 0.5 : -0.5));
                        t = k;
                        hi = x - t * ln2_hi;    /* t*ln2_hi is exact here */
                        lo = t * ln2_lo;
                }

#endif /* ! codereview */
                x = hi - lo;
                c = (hi - x) - lo;      /* still at |x| > 0.5 ln2 */
        } else if (hx < 0x3c900000) {
                /* when |x|<2**-54, return x */
                t = huge + x;           /* return x w/inexact when x != 0 */
                return (x - (t - (huge + x)));
        } else {
        } else
                /* |x| <= 0.5 ln2 */
                k = 0;
        }
```

```
#endif /* ! codereview */

        /* x is now in primary range */
        hfx = 0.5 * x;
        hxs = x * hfx;
        r1 = one + hxs * (Q1 + hxs * (Q2 + hxs * (Q3 + hxs * (Q4 + hxs * Q5))));
        t = 3.0 - r1 * hfx;
        e = hxs * ((r1 - t) / (6.0 - x * t));

        if (k == 0) {                   /* |x| <= 0.5 ln2 */
        if (k == 0) /* |x| <= 0.5 ln2 */
                return (x - (x * e - hxs));
        } else {                        /* |x| > 0.5 ln2 */
        else {          /* |x| > 0.5 ln2 */
                e = (x * (e - c) - c);
                e -= hxs;

#endif /* ! codereview */
                if (k == -1)
                        return (0.5 * (x - e) - 0.5);

#endif /* ! codereview */
                if (k == 1) {
                        if (x < -0.25)
                                return (-2.0 * (e - (x + 0.5)));
                        else
                                return (one + 2.0 * (x - e));
                }

#endif /* ! codereview */
                if (k <= -2 || k > 56) {        /* suffice to return exp(x)-1 */
                        y = one - (e - x);
                        ((int *)&y)[HIWORD] += k << 20;
                        ((int *) &y)[HIWORD] += k << 20;
                        return (y - one);
                }

#endif /* ! codereview */
                t = one;

#endif /* ! codereview */
                if (k < 20) {
                        ((int *)&t)[HIWORD] = 0x3ff00000 - (0x200000 >> k);
                        ((int *) &t)[HIWORD] = 0x3ff00000 - (0x200000 >> k);
                        /* t = 1 - 2^-k */
                        y = t - (e - x);
                        ((int *)&y)[HIWORD] += k << 20;
                        ((int *) &y)[HIWORD] += k << 20;
                } else {
                        ((int *)&t)[HIWORD] = (0x3ff - k) << 20; /* 2^-k */
                        ((int *) &t)[HIWORD] = (0x3ff - k) << 20; /* 2^-k */
                        y = x - (e + t);
                        y += one;
                        ((int *)&y)[HIWORD] += k << 20;
                        ((int *) &y)[HIWORD] += k << 20;
                }
        }

#endif /* ! codereview */
        return (y);
}
```

```
**********************************************************
    1178 Thu Jun  6 20:37:46 2019
new/usr/src/lib/libm/common/C/fabs.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __fabs = fabs

#include "libm.h"
#include "libm_macros.h"
#include <math.h>

double
fabs(double x)
{
        int *px = (int *)&x;
fabs(double x) {
        int *px = (int *) &x;

        px[HIWORD] &= ~0x80000000;
        return (x);
}
______unchanged_portion_omitted_
```

```
**********************************************************
    1736 Thu Jun  6 20:37:47 2019
new/usr/src/lib/libm/common/C/floor.c
11210 libm should be cstyle(10NBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

#endif /* ! codereview */
/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2005 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __floor = floor

/*
 * floor(x) returns the biggest integral value less than or equal to x.
 * NOTE: floor(x) returns result with the same sign as x's, including 0.
 *
 * Modified 8/4/04 for performance.
 */

#include "libm.h"

static const double zero = 0.0, one = 1.0, two52 = 4503599627370496.0;
static const double
        zero = 0.0,
        one = 1.0,
        two52 = 4503599627370496.0;

double
floor(double x)
{
floor(double x) {
        double t, w;
        int hx, lx, ix;

        hx = ((int *)&x)[HIWORD];
        lx = ((int *)&x)[LOWORD];
        ix = hx & ~0x80000000;

#endif /* ! codereview */
```



```
        if (ix >= 0x43300000)   /* return x if |x| >= 2^52, or x is NaN */
                return (x * one);

        t = (hx >= 0) ? two52 : -two52;
        t = (hx >= 0)? two52 : -two52;
        w = x + t;
        t = w - t;

#endif /* ! codereview */
        if (ix < 0x3ff00000) {
                if ((ix | lx) == 0)
                        return (x);
                else
                        return ((hx < 0) ? -one : zero);
                        return ((hx < 0)? -one : zero);
        }

        return ((t <= x) ? t : t - one);
        return ((t <= x)? t : t - one);
}
______unchanged_portion_omitted_
```

```
**********************************************************
    3222 Thu Jun  6 20:37:47 2019
new/usr/src/lib/libm/common/C/fmod.c
11210 libm should be cstyle(10NBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

#endif /* ! codereview */
/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __fmod = fmod

#include "libm.h"

static const double zero = 0.0;

/*
 * The following implementation assumes fast 64-bit integer arith-
 * metic.  This is fine for sparc because we build libm in v8plus
 * mode.  It's also fine for sparcv9 and amd64, although we have
 * assembly code on amd64.  For x86, it would be better to use
 * 32-bit code, but we have assembly for x86, too.
 */
double
fmod(double x, double y)
{
fmod(double x, double y) {
        double w;
        long long hx, ix, iy, iz;
        int nd, k, ny;

        hx = *(long long *)&x;
        ix = hx & ~0x8000000000000000ull;
        iy = *(long long *)&y & ~0x8000000000000000ull;

        /* handle special cases */
        if (iy == 0ll)
                return (_SVID_libm_err(x, y, 27));
```



```
        if (ix >= 0x7ff0000000000000ll || iy > 0x7ff0000000000000ll)
                return ((x * y) * zero);

        if (ix <= iy)
                return ((ix < iy) ? x : x *zero);
                return ((ix < iy)? x : x * zero);

        /*
         * Set:
         *      ny = true exponent of y
         *      nd = true exponent of x minus true exponent of y
         *      ix = normalized significand of x
         *      iy = normalized significand of y
         */
        ny = iy >> 52;
        k = ix >> 52;

#endif /* ! codereview */
        if (ny == 0) {
                /* y is subnormal, x could be normal or subnormal */
                ny = 1;

#endif /* ! codereview */
                while (iy < 0x0010000000000000ll) {
                        ny -= 1;
                        iy += iy;
                }

#endif /* ! codereview */
                nd = k - ny;

#endif /* ! codereview */
                if (k == 0) {
                        nd += 1;

#endif /* ! codereview */
                        while (ix < 0x0010000000000000ll) {
                                nd -= 1;
                                ix += ix;
                        }
                } else {
                        ix = 0x0010000000000000ll | (ix & 0x000fffffffffffffll);
                }
        } else {
                /* both x and y are normal */
                nd = k - ny;
                ix = 0x0010000000000000ll | (ix & 0x000fffffffffffffll);
                iy = 0x0010000000000000ll | (iy & 0x000fffffffffffffll);
        }

        /* perform fixed point mod */
        while (nd--) {
                iz = ix - iy;

#endif /* ! codereview */
                if (iz >= 0)
                        ix = iz;

#endif /* ! codereview */
                ix += ix;
        }

#endif /* ! codereview */
        iz = ix - iy;

#endif /* ! codereview */
```

```
        if (iz >= 0)
                ix = iz;

        /* convert back to floating point and restore the sign */
        if (ix == 0ll)
                return (x * zero);

#endif /* ! codereview */
        while (ix < 0x0010000000000000ll) {
                ix += ix;
                ny -= 1;
        }

#endif /* ! codereview */
        while (ix > 0x0020000000000000ll) {     /* XXX can this ever happen? */
                ny += 1;
                ix >>= 1;
        }

#endif /* ! codereview */
        if (ny <= 0) {
                /* result is subnormal */
                k = -ny + 1;
                ix >>= k;
                *(long long *) &w = (hx & 0x8000000000000000ull) | ix;
                *(long long *)&w = (hx & 0x8000000000000000ull) | ix;
                return (w);
        }

        *(long long *) &w = (hx & 0x8000000000000000ull) | ((long long)ny <<
            52) | (ix & 0x000fffffffffffffll);
        *(long long *)&w = (hx & 0x8000000000000000ull) |
            ((long long)ny << 52) | (ix & 0x000fffffffffffffll);
        return (w);
}
```

_____*unchanged_portion_omitted_*

```
**********************************************************
    1335 Thu Jun  6 20:37:48 2019
new/usr/src/lib/libm/common/C/gamma.c
11210 libm should be cstyle(10NBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

#endif /* ! codereview */
/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2005 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __gamma = gamma

#include "libm.h"

extern int signgam;

double
gamma(double x)
{
gamma(double x) {
        double g;

        if (!finite(x))
                return (x * x);

        g = rint(x);

#endif /* ! codereview */
        if (x == g && x <= 0.0) {
                signgam = 1;
                return (_SVID_libm_err(x, x, 41));
        }

        g = __k_lgamma(x, &signgam);

#endif /* ! codereview */
        if (!finite(g))
                g = _SVID_libm_err(x, x, 40);
```



```
#endif /* ! codereview */
        return (g);
}
```

```
**********************************************************
    1128 Thu Jun  6 20:37:48 2019
new/usr/src/lib/libm/common/C/gamma_r.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

#endif /* ! codereview */
/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2005 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __gamma_r = gamma_r

#include "libm.h"

double
gamma_r(double x, int *signgamp)
{
gamma_r(double x, int *signgamp) {
        return (lgamma_r(x, signgamp));
}
______unchanged_portion_omitted_
```

```
**********************************************************
    5487 Thu Jun  6 20:37:49 2019
new/usr/src/lib/libm/common/C/hypot.c
11210 libm should be cstyle(10NBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __hypot = hypot


/* INDENT OFF */
/*
 * Hypot(x, y)
 * by K.C. Ng for SUN 4.0 libm, updated 3/11/2003.
 * Method :
 * A. When rounding is rounded-to-nearest:
 *      If z = x * x + y * y has error less than sqrt(2) / 2 ulp than
 *      sqrt(z) has error less than 1 ulp.
 *      So, compute sqrt(x*x+y*y) with some care as follows:
 *      Assume x > y > 0;
 *      1. Check whether save and set rounding to round-to-nearest
 *      2. if x > 2y  use
 *              xh*xh+(y*y+((x-xh)*(x+xh))) for x*x+y*y
 *      where xh = x with lower 32 bits cleared;  else
 *      3. if x <= 2y use
 *              x2h*yh+((x-y)*(x-y)+(x2h*(y-yh)+(x2-x2h)*y))
 *      where x2 = 2*x, x2h = 2x with lower 32 bits cleared, yh = y with
 *      lower 32 bits chopped.
 *
 * B. When rounding is not rounded-to-nearest:
 *      The following (magic) formula will yield an error less than 1 ulp.
 *      z = sqrt(x * x + y * y)
 *              hypot(x, y) = x + (y / ((x + z) / y))
 *
 * NOTE: DO NOT remove parenthsis!
 *
 * Special cases:
```



```
 *      hypot(x, y) is INF if x or y is +INF or -INF; else
 *      hypot(x, y) is NAN if x or y is NAN.
 *
 * Accuracy:
 *      hypot(x, y) returns sqrt(x^2+y^2) with error less than 1 ulps
 *      (units in the last place)
 */

#include "libm.h"

static const double zero = 0.0,
static const double
        zero = 0.0,
        onep1u = 1.00000000000000022204e+00,    /* 0x3ff00000 1 = 1+2**-52 */
        twom53 = 1.11022302462515654042e-16,    /* 0x3ca00000 0 = 2**-53 */
        twom768 = 6.441148769597133308e-232,    /* 2^-768 */
        two768 = 1.552518092300708935e+231;     /* 2^768 */

/* INDENT ON */

double
hypot(double x, double y)
{
hypot(double x, double y) {
        double xh, yh, w, ax, ay;
        int i, j, nx, ny, ix, iy, iscale = 0;
        unsigned lx, ly;

        ix = ((int *)&x)[HIWORD] & ~0x80000000;
        lx = ((int *)&x)[LOWORD];
        iy = ((int *)&y)[HIWORD] & ~0x80000000;
        ly = ((int *)&y)[LOWORD];

        ix = ((int *) &x)[HIWORD] & ~0x80000000;
        lx = ((int *) &x)[LOWORD];
        iy = ((int *) &y)[HIWORD] & ~0x80000000;
        ly = ((int *) &y)[LOWORD];
/*
 * Force ax = |x| ~>~ ay = |y|
 */
        if (iy > ix) {
                ax = fabs(y);
                ay = fabs(x);
                i = ix;
                ix = iy;
                iy = i;
                i = lx;
                lx = ly;
                ly = i;
        } else {
                ax = fabs(x);
                ay = fabs(y);
        }

#endif /* ! codereview */
        nx = ix >> 20;
        ny = iy >> 20;
        j = nx - ny;

#endif /* ! codereview */
/*
 * x >= 2^500 (x*x or y*y may overflow)
 */
        if (nx >= 0x5f3) {
                if (nx == 0x7ff) {      /* inf or NaN, signal of sNaN */
                        if (((ix - 0x7ff00000) | lx) == 0)
```

```
                                return (ax == ay ? ay : ax);
                        else if (((iy - 0x7ff00000) | ly) == 0)
                                return (ay == ax ? ax : ay);
                        else
                                return (ax * ay);       /* + -> * for Cheetah */
                } else if (j > 32) {                    /* x >> y */
                        if (j <= 53)
                                ay *= twom53;

#endif /* ! codereview */
                        ax += ay;

                        if (((int *)&ax)[HIWORD] == 0x7ff00000)
                        if (((int *) &ax)[HIWORD] == 0x7ff00000)
                                ax = _SVID_libm_err(x, y, 4);

#endif /* ! codereview */
                        return (ax);
                }

#endif /* ! codereview */
                ax *= twom768;
                ay *= twom768;
                iscale = 2;
                ix -= 768 << 20;
                iy -= 768 << 20;
        }

#endif /* ! codereview */
/*
 * y < 2^-450 (x*x or y*y may underflow)
 */
        else if (ny < 0x23d) {
                if ((ix | lx) == 0)
                        return (ay);

#endif /* ! codereview */
                if ((iy | ly) == 0)
                        return (ax);

#endif /* ! codereview */
                if (j > 53)             /* x >> y */
                        return (ax + ay);

#endif /* ! codereview */
                iscale = 1;
                ax *= two768;
                ay *= two768;

#endif /* ! codereview */
                if (nx == 0) {
                        if (ax == zero) /* guard subnormal flush to zero */
                                return (ax);

                        ix = ((int *)&ax)[HIWORD];
                } else {
                        ix = ((int *) &ax)[HIWORD];
                } else
                        ix += 768 << 20;
                }

#endif /* ! codereview */
                if (ny == 0) {
                        if (ay == zero) /* guard subnormal flush to zero */
                                return (ax * twom768);
```



```
                        iy = ((int *)&ay)[HIWORD];
                } else {
                        iy = ((int *) &ay)[HIWORD];
                } else
                        iy += 768 << 20;
                }

#endif /* ! codereview */
                j = (ix >> 20) - (iy >> 20);

#endif /* ! codereview */
                if (j > 32) {           /* x >> y */
                        if (j <= 53)
                                ay *= twom53;

#endif /* ! codereview */
                        return ((ax + ay) * twom768);
                }
        } else if (j > 32) {            /* x >> y */
                if (j <= 53)
                        ay *= twom53;

#endif /* ! codereview */
                return (ax + ay);
        }

#endif /* ! codereview */
/*
 * Medium range ax and ay with max{|ax/ay|,|ay/ax|} bounded by 2^32
 * First check rounding mode by comparing onep1u*onep1u with onep1u+twom53.
 * Make sure the computation is done at run-time.
 */
        if (((lx | ly) << 5) == 0) {
                ay = ay * ay;
                ax += ay / (ax + sqrt(ax * ax + ay));
        } else if (onep1u * onep1u != onep1u + twom53) {
                /*
                 * round-to-zero, positive, negative mode
                 * magic formula with less than an ulp error
                 */
        } else
        if (onep1u * onep1u != onep1u + twom53) {
        /* round-to-zero, positive, negative mode */
        /* magic formula with less than an ulp error */
                w = sqrt(ax * ax + ay * ay);
                ax += ay / ((ax + w) / ay);
        } else {
                /* round-to-nearest mode */
                w = ax - ay;

#endif /* ! codereview */
                if (w > ay) {
                        ((int *)&xh)[HIWORD] = ix;
                        ((int *)&xh)[LOWORD] = 0;
                        ((int *) &xh)[HIWORD] = ix;
                        ((int *) &xh)[LOWORD] = 0;
                        ay = ay * ay + (ax - xh) * (ax + xh);
                        ax = sqrt(xh * xh + ay);
                } else {
                        ax = ax + ax;
                        ((int *)&xh)[HIWORD] = ix + 0x00100000;
                        ((int *)&xh)[LOWORD] = 0;
                        ((int *)&yh)[HIWORD] = iy;
                        ((int *)&yh)[LOWORD] = 0;
                        ((int *) &xh)[HIWORD] = ix + 0x00100000;
                        ((int *) &xh)[LOWORD] = 0;
```

```
122                         ((int *) &yh)[HIWORD] = iy;
123                         ((int *) &yh)[LOWORD] = 0;
238                         ay = w * w + ((ax - xh) * yh + (ay - yh) * ax);
239                         ax = sqrt(xh * yh + ay);
240                 }
241         }

243 #endif /* ! codereview */
244         if (iscale > 0) {
245                 if (iscale == 1) {
128                 if (iscale == 1)
246                         ax *= twom768;
247                 } else {
130                 else {
248                         ax *= two768;   /* must generate side effect here */

250                         if (((int *)&ax)[HIWORD] == 0x7ff00000)
132                         if (((int *) &ax)[HIWORD] == 0x7ff00000)
251                                 ax = _SVID_libm_err(x, y, 4);
252                 }
253         }

255 #endif /* ! codereview */
256         return (ax);
257 }
```

```
**********************************************************
    2254 Thu Jun  6 20:37:49 2019
new/usr/src/lib/libm/common/C/ilogb.c
11210 libm should be cstyle(10NBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __ilogb = ilogb

#include "libm.h"
#include "xpg6.h"                       /* __xpg6 */

#if defined(__x86)
static const double two52 = 4503599627370496.0;
#else
/*
 * v: high part of a non-zero subnormal |x|; w: low part of |x|
 */
static int
ilogb_subnormal(unsigned v, unsigned w)
{
ilogb_subnormal(unsigned v, unsigned w) {
        int r = -1022 - 52;

        if (v)
                r += 32;
        else
                v = w;

#endif /* ! codereview */
        if (v & 0xffff0000)
                r += 16, v >>= 16;

#endif /* ! codereview */
        if (v & 0xff00)
                r += 8, v >>= 8;
```



```
#endif /* ! codereview */
        if (v & 0xf0)
                r += 4, v >>= 4;

#endif /* ! codereview */
        v <<= 1;
        return (r + ((0xffffaa50 >> v) & 0x3));
}
#endif /* defined(__x86) */

static int
raise_invalid(int v)    /* SUSv3 requires ilogb(0,+/-Inf,NaN) raise invalid */
{
raise_invalid(int v) {  /* SUSv3 requires ilogb(0,+/-Inf,NaN) raise invalid */
#ifndef lint
        if ((__xpg6 & _C99SUSv3_ilogb_0InfNaN_raises_invalid) != 0) {
                static const double zero = 0.0;
                volatile double dummy __unused;

                dummy = zero / zero;
        }
#endif
        return (v);
}

int
ilogb(double x)
{
        int *px = (int *)&x, k = px[HIWORD] & ~0x80000000;
ilogb(double x) {
        int *px = (int *) &x, k = px[HIWORD] & ~0x80000000;

        if (k < 0x00100000) {
                if ((px[LOWORD] | k) == 0) {
                if ((px[LOWORD] | k) == 0)
                        return (raise_invalid(0x80000001));
                } else {
                else {
#if defined(__x86)
                        x *= two52;
                        return (((px[HIWORD] & 0x7ff00000) >> 20) - 1075);
#else
                        return (ilogb_subnormal(k, px[LOWORD]));
#endif
                }
        } else if (k < 0x7ff00000) {
        } else if (k < 0x7ff00000)
                return ((k >> 20) - 1023);
        } else {
        else
                return (raise_invalid(0x7fffffff));
        }
#endif /* ! codereview */
}
```

```
**********************************************************
    1567 Thu Jun  6 20:37:50 2019
new/usr/src/lib/libm/common/C/isnan.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

#endif /* ! codereview */
/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak isnan = __isnan
#pragma weak _isnan = __isnan
#pragma weak _isnand = __isnan
#pragma weak isnand = __isnan

#include "libm.h"

/*
 * The following implementation assumes fast 64-bit integer arith-
 * metic.  This is fine for sparc because we build libm in v8plus
 * mode.  It’s also fine for sparcv9 and amd64.  For x86, it would
 * be better to use 32-bit code, but we have assembly for x86.
 */
int
__isnan(double x)
{
__isnan(double x) {
        long long llx;

        llx = *(long long *)&x & ~0x8000000000000000ull;
        return ((unsigned long long)(0x7ff0000000000000ll - llx) >> 63);
}
______unchanged_portion_omitted_
```

```
**********************************************************
    9073 Thu Jun  6 20:37:50 2019
new/usr/src/lib/libm/common/C/j0.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

/*
 * Floating point Bessel's function of the first and second kinds
 * of order zero: j0(x),y0(x);
 *
 * Special cases:
 *      y0(0)=y1(0)=yn(n,0) = -inf with division by zero signal;
 *      y0(-ve)=y1(-ve)=yn(n,-ve) are NaN with invalid signal.
 */

#pragma weak __j0 = j0
#pragma weak __y0 = y0

#include "libm.h"
#include "libm_protos.h"
#include <math.h>
#include <values.h>

#define GENERIC double
static const GENERIC
zero    = 0.0,
small   = 1.0e-5,
tiny    = 1.0e-18,
one     = 1.0,
eight   = 8.0,
invsqrtpi = 5.641895835477562869480794515607725858441e-0001,
tpi     = 0.636619772367581343075535053490057448;

static const GENERIC zero = 0.0,
        small = 1.0e-5,
        tiny = 1.0e-18,
```



```
        one = 1.0,
        eight = 8.0,
        invsqrtpi = 5.641895835477562869480794515607725858441e-0001,
        tpi = 0.636619772367581343075535053490057448;

static GENERIC pzero(GENERIC);
static GENERIC qzero(GENERIC);

static GENERIC pzero(GENERIC), qzero(GENERIC);
static const GENERIC r0[4] = {          /* [1.e-5, 1.28] */
        -2.500000000000003622131880894830476755537e-0001,
        1.095597547334830263234433855932375353303e-0002,
        -1.819734750463320921799187258987098087697e-0004,
        9.977001946806131657544212501069893930846e-0007,
};

#endif /* ! codereview */
static const GENERIC s0[4] = {          /* [1.e-5, 1.28] */
        1.0,
        1.867609810662950169966782360588199673741e-0002,
        1.590389206181565490878430827706972074208e-0004,
        6.520867386742583632375520147714499522721e-0007,
};

#endif /* ! codereview */
static const GENERIC r1[9] = {          /* [1.28,8] */
        9.999999999999999942156495584397047660949e-0001,
        -2.389887722731319130476839836908143731281e-0001,
        1.293359476138939027791270393439493640570e-0002,
        -2.770985642343140122168852400228563364082e-0004,
        2.905241575772067678086738389169625218912e-0006,
        -1.636846356264052597969042009265043251279e-0008,
        5.072306160724884775085431059052611737827e-0011,
        -8.187060730684066824228914775146536139112e-0014,
        5.422219326959949863954297860723723423842e-0017,
};

#endif /* ! codereview */
static const GENERIC s1[9] = {          /* [1.28,8] */
        1.0,
        1.101122772686807702762104741932076228349e-0002,
        6.140169310641649223411427764669143978228e-0005,
        2.292035877515152097976946119293215705250e-0007,
        6.356910426504644334558832036362219583789e-0010,
        1.366626326900219555045096999553948891401e-0012,
        2.280399586866739522891837985560481180088e-0015,
        2.801559820648939665270492520004836611187e-0018,
        2.073101088320349159764410261466350732968e-0021,
};

GENERIC
j0(GENERIC x)
{
j0(GENERIC x) {
        GENERIC z, s, c, ss, cc, r, u, v, ox;
        int i;

        if (isnan(x))
                return (x * x);         /* + -> * for Cheetah */

                return (x*x);   /* + -> * for Cheetah */
        ox = x;
        x = fabs(x);

#endif /* ! codereview */
        if (x > 8.0) {
```

```
                if (!finite(x))
                        return (zero);

#endif /* ! codereview */
                s = sin(x);
                c = cos(x);

                /* BEGIN CSTYLED */
#endif /* ! codereview */
                /*
                 * j0(x) = sqrt(2/(pi*x))*(p0(x)*cos(x0)-q0(x)*sin(x0))
                 * where x0 = x-pi/4
                 *      Better formula:
                 *              cos(x0) = cos(x)cos(pi/4)+sin(x)sin(pi/4)
                 *                      = 1/sqrt(2) * (cos(x) + sin(x))
                 *              sin(x0) = sin(x)cos(pi/4)-cos(x)sin(pi/4)
                 *                      = 1/sqrt(2) * (sin(x) - cos(x))
                 * To avoid cancellation, use
                 *              sin(x) +- cos(x) = -cos(2x)/(sin(x) -+ cos(x))
                 * to compute the worse one.
                 */
                /* END CSTYLED */
#endif /* ! codereview */
                if (x > 8.9e307) {      /* x+x may overflow */
                        ss = s - c;
                        cc = s + c;
                        ss = s-c;
                        cc = s+c;
                } else if (signbit(s) != signbit(c)) {
                        ss = s - c;
                        cc = -cos(x + x) / ss;
                        cc = -cos(x+x)/ss;
                } else {
                        cc = s + c;
                        ss = -cos(x + x) / cc;
                        ss = -cos(x+x)/cc;
                }

#endif /* ! codereview */
                /*
                 * j0(x) = 1/sqrt(pi) * (P(0,x)*cc - Q(0,x)*ss) / sqrt(x)
                 * y0(x) = 1/sqrt(pi) * (P(0,x)*ss + Q(0,x)*cc) / sqrt(x)
                 */
                if (x > 1.0e40) {
                        z = (invsqrtpi * cc) / sqrt(x);
                } else {
                        u = pzero(x);
                        v = qzero(x);
                        z = invsqrtpi * (u * cc - v * ss) / sqrt(x);
                if (x > 1.0e40) z = (invsqrtpi*cc)/sqrt(x);
                else {
                    u = pzero(x); v = qzero(x);
                    z = invsqrtpi*(u*cc-v*ss)/sqrt(x);
                }

#endif /* ! codereview */
                /* force to pass SVR4 even the result is wrong (sign) */
                if (x > X_TLOSS)
                        return (_SVID_libm_err(ox, z, 34));
                else
                        return (z);
        }

#endif /* ! codereview */
        if (x <= small) {
                if (x <= tiny)
```



```
                        return (one - x);
                        return (one-x);
                else
                        return (one - x * x * 0.25);
                        return (one-x*x*0.25);
        }

        z = x * x;

        z = x*x;
        if (x <= 1.28) {
                r = r0[0] + z * (r0[1] + z * (r0[2] + z * r0[3]));
                s = s0[0] + z * (s0[1] + z * (s0[2] + z * s0[3]));
                return (one + z * (r / s));
            r =  r0[0]+z*(r0[1]+z*(r0[2]+z*r0[3]));
            s =  s0[0]+z*(s0[1]+z*(s0[2]+z*s0[3]));
            return (one + z*(r/s));
        } else {
                for (r = r1[8], s = s1[8], i = 7; i >= 0; i--) {
                        r = r * z + r1[i];
                        s = s * z + s1[i];
                r = r*z + r1[i];
                s = s*z + s1[i];
                }

                return (r / s);
            return (r/s);
        }
}
```

*_____unchanged_portion_omitted_*

```
#endif /* ! codereview */
static const GENERIC v0[5] = {
        1.0,
        4.67867893151254900258770247734921488647 5e-0003,
        9.48682895552994853482280082949756517898 5e-0006,
        1.00149592915886164665901084413668245490 6e-0008,
        4.72533811625602166020444323568535859361 1e-0012,
};

GENERIC
y0(GENERIC x)
{
y0(GENERIC x) {
        GENERIC z, /* d, */ s, c, ss, cc, u, v;
        int i;

        if (isnan(x))
                return (x * x);         /* + -> * for Cheetah */

                return (x*x);   /* + -> * for Cheetah */
        if (x <= zero) {
                if (x == zero)
                        /* d= -one/(x-x); */
                        return (_SVID_libm_err(x, x, 8));
                else
                        /* d = zero/(x-x); */
                        return (_SVID_libm_err(x, x, 9));
        }

#endif /* ! codereview */
        if (x > 8.0) {
                if (!finite(x))
                        return (zero);

#endif /* ! codereview */
```

```
                s = sin(x);
                c = cos(x);

                /* BEGIN CSTYLED */
#endif /* ! codereview */
                /*
                 * j0(x) = sqrt(2/(pi*x))*(p0(x)*cos(x0)-q0(x)*sin(x0))
                 * where x0 = x-pi/4
                 *      Better formula:
                 *              cos(x0) = cos(x)cos(pi/4)+sin(x)sin(pi/4)
                 *                      = 1/sqrt(2) * (cos(x) + sin(x))
                 *              sin(x0) = sin(x)cos(pi/4)-cos(x)sin(pi/4)
                 *                      = 1/sqrt(2) * (sin(x) - cos(x))
                 * To avoid cancellation, use
                 *              sin(x) +- cos(x) = -cos(2x)/(sin(x) -+ cos(x))
                 * to compute the worse one.
                 */
                /* END CSTYLED */
#endif /* ! codereview */
                if (x > 8.9e307) {      /* x+x may overflow */
                        ss = s - c;
                        cc = s + c;
                        ss = s-c;
                        cc = s+c;
                } else if (signbit(s) != signbit(c)) {
                        ss = s - c;
                        cc = -cos(x + x) / ss;
                        cc = -cos(x+x)/ss;
                } else {
                        cc = s + c;
                        ss = -cos(x + x) / cc;
                        ss = -cos(x+x)/cc;
                }

#endif /* ! codereview */
                /*
                 * j0(x) = 1/sqrt(pi*x) * (P(0,x)*cc - Q(0,x)*ss)
                 * y0(x) = 1/sqrt(pi*x) * (P(0,x)*ss + Q(0,x)*cc)
                 */
                if (x > 1.0e40)
                        z = (invsqrtpi * ss) / sqrt(x);
                    z = (invsqrtpi*ss)/sqrt(x);
                else
                        z = invsqrtpi * (pzero(x) * ss + qzero(x) * cc) /
                            sqrt(x);

                    z =  invsqrtpi*(pzero(x)*ss+qzero(x)*cc)/sqrt(x);
                if (x > X_TLOSS)
                        return (_SVID_libm_err(x, z, 35));
                else
                        return (z);
        }

        if (x <= tiny)
                return (u0[0] + tpi * log(x));

        z = x * x;

        for (u = u0[12], i = 11; i >= 0; i--)
                u = u * z + u0[i];
#endif /* ! codereview */

        v = v0[0] + z * (v0[1] + z * (v0[2] + z * (v0[3] + z * v0[4])));
        return (u / v + tpi * (j0(x) * log(x)));
        }
        if (x <= tiny) {
```



```
            return (u0[0] + tpi*log(x));
        }
        z = x*x;
        for (u = u0[12], i = 11; i >= 0; i--) u = u*z + u0[i];
        v = v0[0]+z*(v0[1]+z*(v0[2]+z*(v0[3]+z*v0[4])));
        return (u/v + tpi*(j0(x)*log(x)));
}

static const GENERIC pr[7] = {          /* [8 -- inf]  pzero 6550 */
        .486134418338605272139123844 7e5, .137766254940711227813343894 5e6,
        .122246636408828973186911400 4e6, .410707008431517613558335537 4e5,
        .502607380186063712588903991 5e4, .178319365912547965454154241 9e3,
        .486134418338605272139123844 7e5,
        .137766254940711227813343894 5e6,
        .122246636408828973186911400 4e6,
        .410707008431517613558335537 4e5,
        .502607380186063712588903991 5e4,
        .178319365912547965454154241 9e3,
        .880103440553834216916775 64e0,
};

#endif /* ! codereview */
static const GENERIC ps[7] = {          /* [8 -- inf] pzero 6550 */
        .486134418338605272141403705 8e5, .137819663263038467047758269 9e6,
        .122396718534100654274893678 7e6, .412015024379535363999586261 7e5,
        .506827118105354639249018435 3e4, .182981790547276996053567166 4e3,
        .486134418338605272141403705 8e5,
        .137819663263038467047758269 9e6,
        .122396718534100654274893678 7e6,
        .412015024379535363999586261 7e5,
        .506827118105354639249018435 3e4,
        .182981790547276996053567166 4e3,
        1.0,
};

#endif /* ! codereview */
static const GENERIC huge = 1.0e10;

static GENERIC
pzero(GENERIC x)
{
pzero(GENERIC x) {
        GENERIC s, r, t, z;
        int i;

#endif /* ! codereview */
        if (x > huge)
                return (one);

        t = eight / x;
        z = t * t;
        r = pr[5] + z * pr[6];
        s = ps[5] + z;

        t = eight/x; z = t*t;
        r = pr[5]+z*pr[6];
        s = ps[5]+z;
        for (i = 4; i >= 0; i--) {
                r = r * z + pr[i];
                s = s * z + ps[i];
            r = r*z + pr[i];
            s = s*z + ps[i];
        }

        return (r / s);
        return (r/s);
```

```
}

static const GENERIC qr[7] = {          /* [8 -- inf]  qzero 6950 */
        -.173121099570106853918561195 1e3, -.552255916593616696123524061 3e3,
        -.560493560663734659061452961 3e3, -.220043030022600937947736501 1e3,
        -.323869355375648849771296746e2, -.142949792079079562234992581e1,
        -.173121099570106853918561195 1e3,
        -.552255916593616696123524061 3e3,
        -.560493560663734659061452961 3e3,
        -.220043030022600937947736501 1e3,
        -.323869355375648849771296746e2,
        -.142949792079079562234992581e1,
        -.834690374102384988158918e-2,
};

#endif /* ! codereview */
static const GENERIC qs[7] = {          /* [8 -- inf] qzero 6950 */
        .110797503724868386532670964 5e5, .354458168062708267465147187 3e5,
        .361911893791839413217901905 9e5, .143989556356539800747148582 2e5,
        .219027702334436395593022623 4e4, .106695157020407986137501682e3,
        .110797503724868386532670964 5e5,
        .354458168062708267465147187 3e5,
        .361911893791839413217901905 9e5,
        .143989556356539800747148582 2e5,
        .219027702334436395593022623 4e4,
        .106695157020407986137501682e3,
        1.0,
};

static GENERIC
qzero(GENERIC x)
{
qzero(GENERIC x) {
        GENERIC s, r, t, z;
        int i;

#endif /* ! codereview */
        if (x > huge)
                return (-0.125 / x);

        t = eight / x;
        z = t * t;
        r = qr[5] + z * qr[6];
        s = qs[5] + z;

                return (-0.125/x);
        t = eight/x; z = t*t;
        r = qr[5]+z*qr[6];
        s = qs[5]+z;
        for (i = 4; i >= 0; i--) {
                r = r * z + qr[i];
                s = s * z + qs[i];
            r = r*z + qr[i];
            s = s*z + qs[i];
        }

        return (t * (r / s));
        return (t*(r/s));
}
```

_____*unchanged_portion_omitted_*

```
**********************************************************
    9463 Thu Jun  6 20:37:51 2019
new/usr/src/lib/libm/common/C/j1.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

/*
 * floating point Bessel's function of the first and second kinds
 * of order zero: j1(x),y1(x);
 *
 * Special cases:
 *      y0(0)=y1(0)=yn(n,0) = -inf with division by zero signal;
 *      y0(-ve)=y1(-ve)=yn(n,-ve) are NaN with invalid signal.
 */

#pragma weak __j1 = j1
#pragma weak __y1 = y1

#include "libm.h"
#include "libm_protos.h"
#include <math.h>
#include <values.h>

#define GENERIC double
static const GENERIC
zero    = 0.0,
small   = 1.0e-5,
tiny    = 1.0e-20,
one     = 1.0,
invsqrtpi = 5.641895835477562869480794515607725858441e-0001,
tpi     = 0.636619772367581343075535053490057448;

static const GENERIC zero = 0.0,
        small = 1.0e-5,
        tiny = 1.0e-20,
        one = 1.0,
```



```
        invsqrtpi = 5.641895835477562869480794515607725858441e-0001,
        tpi = 0.636619772367581343075535053490057448;

static GENERIC pone(GENERIC);
static GENERIC qone(GENERIC);

static GENERIC pone(GENERIC), qone(GENERIC);
static const GENERIC r0[4] = {
        -6.250000000000002203053200981413218949548e-0002,
        1.600998455640072901321605101981501263762e-0003,
        -1.963888815948313758552511884390162864930e-0005,
        8.263917341093549759781339713418201620998e-0008,
};

#endif /* ! codereview */
static const GENERIC s0[7] = {
        1.0e0,
        1.605069137643004242395356851797873766927e-0002,
        1.149454623251299996428500249509098499383e-0004,
        3.849701673735260970379681807910852327825e-0007,
};

#endif /* ! codereview */
static const GENERIC r1[12] = {
        4.999999999999999995517408894340485471724e-0001,
        -6.003825028120475684835384519945468075423e-0002,
        2.301719899263321828388344461995355419832e-0003,
        -4.208494869238892934859525221654040304068e-0005,
        4.377745135188837783031540029700282443388e-0007,
        -2.854106755678624335145364226735677754179e-0009,
        1.234002865443952024332943901323798413689e-0011,
        -3.645498437039791058951273508838177134310e-0014,
        7.404320596071797459925377103787837414422e-0017,
        -1.009457448277522275262808398517024439084e-0019,
        8.520158355824819796968771418801019930585e-0023,
        -3.458159926081163274483854614601091361424e-0026,
};

#endif /* ! codereview */
static const GENERIC s1[5] = {
        1.0e0,
        4.923499437590484879081138588998986303306e-0003,
        1.054389489212184156499666953501976688452e-0005,
        1.180768373106166527048240364872043816050e-0008,
        5.942665743476099355323245707680648588540e-0012,
};

GENERIC
j1(GENERIC x)
{
j1(GENERIC x) {
        GENERIC z, d, s, c, ss, cc, r;
        int i, sgn;

        if (!finite(x))
                return (one / x);

                return (one/x);
        sgn = signbit(x);
        x = fabs(x);

#endif /* ! codereview */
        if (x > 8.00) {
                s = sin(x);
                c = cos(x);
```

```
                /* BEGIN CSTYLED */
#endif /* ! codereview */
                /*
                 * j1(x) = sqrt(2/(pi*x))*(p1(x)*cos(x0)-q1(x)*sin(x0))
                 * where x0 = x-3pi/4
                 *      Better formula:
                 *              cos(x0) = cos(x)cos(3pi/4)+sin(x)sin(3pi/4)
                 *                      =  1/sqrt(2) * (sin(x) - cos(x))
                 *              sin(x0) = sin(x)cos(3pi/4)-cos(x)sin(3pi/4)
                 *                      = -1/sqrt(2) * (cos(x) + sin(x))
                 * To avoid cancellation, use
                 *              sin(x) +- cos(x) = -cos(2x)/(sin(x) -+ cos(x))
                 * to compute the worse one.
                 */
                /* END CSTYLED */
#endif /* ! codereview */
                if (x > 8.9e307) {      /* x+x may overflow */
                        ss = -s - c;
                        cc = s - c;
                        ss = -s-c;
                        cc =  s-c;
                } else if (signbit(s) != signbit(c)) {
                        cc = s - c;
                        ss = cos(x + x) / cc;
                        ss = cos(x+x)/cc;
                } else {
                        ss = -s - c;
                        cc = cos(x + x) / ss;
                        ss = -s-c;
                        cc = cos(x+x)/ss;
                }

#endif /* ! codereview */
                /*
                 * j1(x) = 1/sqrt(pi*x) * (P(1,x)*cc - Q(1,x)*ss)
                 * y1(x) = 1/sqrt(pi*x) * (P(1,x)*ss + Q(1,x)*cc)
                 */
                if (x > 1.0e40)
                        d = (invsqrtpi * cc) / sqrt(x);
                    d = (invsqrtpi*cc)/sqrt(x);
                else
                        d = invsqrtpi * (pone(x) * cc - qone(x) * ss) / sqrt(x);
                        d =  invsqrtpi*(pone(x)*cc-qone(x)*ss)/sqrt(x);

                if (x > X_TLOSS) {
                        if (sgn != 0) {
                                d = -d;
                                x = -x;
                        }

                    if (sgn != 0) { d = -d; x = -x; }
                        return (_SVID_libm_err(x, d, 36));
                } else if (sgn == 0) {
                } else
                    if (sgn == 0)
                        return (d);
                } else {
                        else
                        return (-d);
                }
        }

#endif /* ! codereview */
        if (x <= small) {
                if (x <= tiny)
                        d = 0.5 * x;
```



```
                        d = 0.5*x;
                else
                        d = x * (0.5 - x * x * 0.125);

                        d =  x*(0.5-x*x*0.125);
                if (sgn == 0)
                        return (d);
                else
                        return (-d);
        }

        z = x * x;

        z = x*x;
        if (x < 1.28) {
                r = r0[3];
                s = s0[3];

#endif /* ! codereview */
                for (i = 2; i >= 0; i--) {
                        r = r * z + r0[i];
                        s = s * z + s0[i];
                r = r*z + r0[i];
                s = s*z + s0[i];
                }

                d = x * 0.5 + x * (z * (r / s));
            d = x*0.5+x*(z*(r/s));
        } else {
                r = r1[11];

                for (i = 10; i >= 0; i--)
                        r = r * z + r1[i];

                s = s1[0] + z * (s1[1] + z * (s1[2] + z * (s1[3] + z * s1[4])));
                d = x * (r / s);
            for (i = 10; i >= 0; i--) r = r*z + r1[i];
            s = s1[0]+z*(s1[1]+z*(s1[2]+z*(s1[3]+z*s1[4]))));
            d = x*(r/s);
        }

#endif /* ! codereview */
        if (sgn == 0)
                return (d);
        else
                return (-d);
}

static const GENERIC u0[4] = {
        -1.960570906462389461018983259589655961560e-0001,
        4.931824118350661953459180060007970291139e-0002,
        -1.626975871565393656845930125424683008677e-0003,
        1.359657517926394132692884168082224258360e-0005,
};

#endif /* ! codereview */
static const GENERIC v0[5] = {
        1.0e0,
        2.565807214838390835108224713630901653793e-0002,
        3.374175208978404268650522752520906231508e-0004,
        2.840368571306070719539936935220728843177e-0006,
        1.396387402048998277638900944415752207592e-0008,
};

#endif /* ! codereview */
static const GENERIC u1[12] = {
```

```
230         -1.960570906462389473336339614647555351626e-0001,
231         5.336268030335074494231369159933012844735e-0002,
232         -2.684137504382748094149184541866332033280e-0003,
233         5.737671618979185736981543498580051903060e-0005,
234         -6.642696350686335339171171785557663224892e-0007,
235         4.692417922568160354012347591960362101664e-0009,
236         -2.161728635907789319335231338621412258355e-0011,
237         6.727353419738316107197644431844194668702e-0014,
238         -1.427502986803861372125234355906790573422e-0016,
239         2.020392498726806769468143219616642940371e-0019,
240         -1.761371948595104156753045457888272716340e-0022,
241         7.352828391941157905175042420249225115816e-0026,
242 };

244 #endif /* ! codereview */
245 static const GENERIC v1[5] = {
246         1.0e0,
247         5.029187436727947764916247076102283399442e-0003,
248         1.102693095808242775074856548927801750627e-0005,
249         1.268035774543174837829534603830227216291e-0008,
250         6.579416271766610825192542295821308730206e-0012,
251 };


253 GENERIC
254 y1(GENERIC x)
255 {
 94 y1(GENERIC x) {
256         GENERIC z, d, s, c, ss, cc, u, v;
257         int i;

259         if (isnan(x))
260                 return (x * x);         /* + -> * for Cheetah */

 99                 return (x*x);   /* + -> * for Cheetah */
262         if (x <= zero) {
263                 if (x == zero)
264                         /* return -one/zero;  */
265                         return (_SVID_libm_err(x, x, 10));
266                 else
267                         /* return zero/zero; */
268                         return (_SVID_libm_err(x, x, 11));
269         }

271 #endif /* ! codereview */
272         if (x > 8.0) {
273                 if (!finite(x))
274                         return (zero);

276 #endif /* ! codereview */
277                 s = sin(x);
278                 c = cos(x);

280                 /* BEGIN CSTYLED */
281 #endif /* ! codereview */
282                 /*
283                  * j1(x) = sqrt(2/(pi*x))*(p1(x)*cos(x0)-q1(x)*sin(x0))
284                  * where x0 = x-3pi/4
285                  *      Better formula:
286                  *              cos(x0) = cos(x)cos(3pi/4)+sin(x)sin(3pi/4)
287                  *                      =  1/sqrt(2) * (sin(x) - cos(x))
288                  *              sin(x0) = sin(x)cos(3pi/4)-cos(x)sin(3pi/4)
289                  *                      = -1/sqrt(2) * (cos(x) + sin(x))
290                  * To avoid cancellation, use
291                  *              sin(x) +- cos(x) = -cos(2x)/(sin(x) -+ cos(x))
292                  * to compute the worse one.
```



```
293                  */
294                 /* END CSTYLED */
295 #endif /* ! codereview */
296                 if (x > 8.9e307) {      /* x+x may overflow */
297                         ss = -s - c;
298                         cc = s - c;
108                         ss = -s-c;
109                         cc =  s-c;
299                 } else if (signbit(s) != signbit(c)) {
300                         cc = s - c;
301                         ss = cos(x + x) / cc;
112                         ss = cos(x+x)/cc;
302                 } else {
303                         ss = -s - c;
304                         cc = cos(x + x) / ss;
114                         ss = -s-c;
115                         cc = cos(x+x)/ss;
305                 }

307 #endif /* ! codereview */
308                 /*
309                  * j1(x) = 1/sqrt(pi*x) * (P(1,x)*cc - Q(1,x)*ss)
310                  * y1(x) = 1/sqrt(pi*x) * (P(1,x)*ss + Q(1,x)*cc)
311                  */
312                 if (x > 1.0e91)
313                         d = (invsqrtpi * ss) / sqrt(x);
117                     d =  (invsqrtpi*ss)/sqrt(x);
314                 else
315                         d = invsqrtpi * (pone(x) * ss + qone(x) * cc) / sqrt(x);
119                         d = invsqrtpi*(pone(x)*ss+qone(x)*cc)/sqrt(x);

317                 if (x > X_TLOSS)
318                         return (_SVID_libm_err(x, d, 37));
319                 else
320                         return (d);
321         }

323         if (x <= tiny)
324                 return (-tpi / x);

326         z = x * x;

126                 if (x <= tiny) {
127                         return (-tpi/x);
128                 }
129         z = x*x;
328         if (x < 1.28) {
329                 u = u0[3];
330                 v = v0[3] + z * v0[4];

131             u = u0[3]; v = v0[3]+z*v0[4];
332                 for (i = 2; i >= 0; i--) {
333                         u = u * z + u0[i];
334                         v = v * z + v0[i];
133                 u = u*z + u0[i];
134                 v = v*z + v0[i];
335                 }
336         } else {
337                 for (u = u1[11], i = 10; i >= 0; i--)
338                         u = u * z + u1[i];

340                 v = v1[0] + z * (v1[1] + z * (v1[2] + z * (v1[3] + z * v1[4])));
137             for (u = u1[11], i = 10; i >= 0; i--) u = u*z+u1[i];
138             v = v1[0]+z*(v1[1]+z*(v1[2]+z*(v1[3]+z*v1[4])));
341         }
```

```
343         return (x * (u / v) + tpi * (j1(x) * log(x) - one / x));
140         return (x*(u/v) + tpi*(j1(x)*log(x)-one/x));
344 }

346 static const GENERIC pr0[6] = {
347         -.443575781679412785711472079 4e7, -.994224650507764119565837789 9e7,
348         -.660337324836493910925524543 4e7, -.152352935118113738325510572 2e7,
349         -.109824055434593467273741313 9e6, -.161161664432461011647741289 8e4,
144         -.443575781679412785711472079 4e7,
145         -.994224650507764119565837789 9e7,
146         -.660337324836493910925524543 4e7,
147         -.152352935118113738325510572 2e7,
148         -.109824055434593467273741313 9e6,
149         -.161161664432461011647741289 8e4,
350 };

352 #endif /* ! codereview */
353 static const GENERIC ps0[6] = {
354         -.443575781679412785682801696 2e7, -.993412438993458565896755630 9e7,
355         -.658533947972308707282691506 9e7, -.151180950663416088164454635 8e7,
356         -.107263859911038201190306386 7e6, -.145500944019049618245356506 8e4,
151         -.443575781679412785682801696 2e7,
152         -.993412438993458565896755630 9e7,
153         -.658533947972308707282691506 9e7,
154         -.151180950663416088164454635 8e7,
155         -.107263859911038201190306386 7e6,
156         -.145500944019049618245356506 8e4,
357 };

359 #endif /* ! codereview */
360 static const GENERIC huge = 1.0e10;

361 static GENERIC
362 pone(GENERIC x)
363 {
160 pone(GENERIC x) {
364         GENERIC s, r, t, z;
365         int i;

367 #endif /* ! codereview */
368         /* assume x > 8 */
369         if (x > huge)
370                 return (one);

372         t = 8.0 / x;
373         z = t * t;
374         r = pr0[5];
375         s = ps0[5] + z;

163         t = 8.0/x; z = t*t;
164         r = pr0[5]; s = ps0[5]+z;
377         for (i = 4; i >= 0; i--) {
378                 r = z * r + pr0[i];
379                 s = z * s + ps0[i];
166                 r = z*r + pr0[i];
167                 s = z*s + ps0[i];
380         }

382         return (r / s);
169         return (r/s);
383 }

385 static const GENERIC qr0[6] = {
386         0.332209134098572235185970444 2e5, 0.851451606753357019655500117 1e5,
387         0.661788365812708351793999216 6e5, 0.184942628732238667965200981 9e5,
```



```
388         0.170637542902076800206128354 6e4, 0.352651338466360321859217558 0e2,
174         0.332209134098572235185970444 2e5,
175         0.851451606753357019655500117 1e5,
176         0.661788365812708351793999216 6e5,
177         0.184942628732238667965200981 9e5,
178         0.170637542902076800206128354 6e4,
179         0.352651338466360321859217558 0e2,
389 };

391 #endif /* ! codereview */
392 static const GENERIC qs0[6] = {
393         0.708712819410287435737750247 2e6, 0.181945804224399729892455383 9e7,
394         0.141946066960372089285575525 3e7, 0.400294435822669751170861081 3e6,
395         0.378902297457722026414295225 6e5, 0.863836776960499096747551718 3e3,
181         0.708712819410287435737750247 2e6,
182         0.181945804224399729892455383 9e7,
183         0.141946066960372089285575525 3e7,
184         0.400294435822669751170861081 3e6,
185         0.378902297457722026414295225 6e5,
186         0.863836776960499096747551718 3e3,
396 };

398 static GENERIC
399 qone(GENERIC x)
400 {
190 qone(GENERIC x) {
401         GENERIC s, r, t, z;
402         int i;

404 #endif /* ! codereview */
405         if (x > huge)
406                 return (0.375 / x);
193                 return (0.375/x);

408         t = 8.0 / x;
409         z = t * t;
195         t = 8.0/x; z = t*t;
410         /* assume x > 8 */
411         r = qr0[5];
412         s = qs0[5] + z;

197         r = qr0[5]; s = qs0[5]+z;
414         for (i = 4; i >= 0; i--) {
415                 r = z * r + qr0[i];
416                 s = z * s + qs0[i];
199                 r = z*r + qr0[i];
200                 s = z*s + qs0[i];
417         }

419         return (t * (r / s));
202         return (t*(r/s));
420 }
```

_____*unchanged_portion_omitted_*

```
**********************************************************
    7660 Thu Jun  6 20:37:51 2019
new/usr/src/lib/libm/common/C/jn.c
11210 libm should be cstyle(10NBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __jn = jn
#pragma weak __yn = yn

/*
 * floating point Bessel's function of the 1st and 2nd kind
 * of order n: jn(n,x),yn(n,x);
 *
 * Special cases:
 *      y0(0)=y1(0)=yn(n,0) = -inf with division by zero signal;
 *      y0(-ve)=y1(-ve)=yn(n,-ve) are NaN with invalid signal.
 * Note 2. About jn(n,x), yn(n,x)
 *      For n=0, j0(x) is called,
 *      for n=1, j1(x) is called,
 *      for n<x, forward recursion us used starting
 *      from values of j0(x) and j1(x).
 *      for n>x, a continued fraction approximation to
 *      j(n,x)/j(n-1,x) is evaluated and then backward
 *      recursion is used starting from a supposed value
 *      for j(n,x). The resulting value of j(0,x) is
 *      compared with the actual value to correct the
 *      supposed value of j(n,x).
 *
 *      yn(n,x) is similar in all respects, except
 *      that forward recursion is used for all
 *      values of n>1.
 *
 */

#include "libm.h"
#include <float.h>                      /* DBL_MIN */
```



```
#include <values.h>                     /* X_TLOSS */
#include "xpg6.h"                       /* __xpg6 */

#define GENERIC double

static const GENERIC
        invsqrtpi = 5.641895835477562869480794515607725858441e-0001,
        two = 2.0,
        zero = 0.0,
        one = 1.0;

GENERIC
jn(int n, GENERIC x)
{
        int i, sgn;
        GENERIC a, b, temp = 0;
        GENERIC z, w, ox, on;

        /*
         * J(-n,x) = (-1)^n * J(n, x), J(n, -x) = (-1)^n * J(n, x)
         * Thus, J(-n,x) = J(n,-x)
         */
        ox = x;
        on = (GENERIC)n;

        if (n < 0) {
                n = -n;
                x = -x;
        }

#endif /* ! codereview */
        if (isnan(x))
                return (x * x);         /* + -> * for Cheetah */

        if (!((int)_lib_version == libm_ieee || (__xpg6 &
            _C99SUSv3_math_errexcept) != 0)) {
                return (x*x);   /* + -> * for Cheetah */
        if (!((int)_lib_version == libm_ieee ||
            (__xpg6 & _C99SUSv3_math_errexcept) != 0)) {
                if (fabs(x) > X_TLOSS)
                        return (_SVID_libm_err(on, ox, 38));
        }

#endif /* ! codereview */
        if (n == 0)
                return (j0(x));

#endif /* ! codereview */
        if (n == 1)
                return (j1(x));

        if ((n & 1) == 0)
        if ((n&1) == 0)
                sgn = 0;                /* even n */
        else
                sgn = signbit(x);       /* old n  */

#endif /* ! codereview */
        x = fabs(x);

        if (x == zero || !finite(x)) {
                b = zero;
        } else if ((GENERIC)n <= x) {
        if (x == zero||!finite(x)) b = zero;
        else if ((GENERIC)n <= x) {
                /*
```

```
 122                  * Safe to use
 123                  *  J(n+1,x)=2n/x *J(n,x)-J(n-1,x)
 124                  */
 125                 if (x > 1.0e91) {
 126                         /*
 127                          * x >> n**2
 128                          *    Jn(x) = cos(x-(2n+1)*pi/4)*sqrt(2/x*pi)
 129                          *   Yn(x) = sin(x-(2n+1)*pi/4)*sqrt(2/x*pi)
 130                          *   Let s=sin(x), c=cos(x),
 131                          *      xn=x-(2n+1)*pi/4, sqt2 = sqrt(2),then
 132                          *
 133                          *         n    sin(xn)*sqt2    cos(xn)*sqt2
 134                          *      ----------------------------------
 135                          *         0     s-c             c+s
 136                          *         1    -s-c            -c+s
 137                          *         2    -s+c            -c-s
 138                          *         3     s+c             c-s
 139                          */
 140                         switch (n & 3) {
  56                         switch (n&3) {
 141                         case 0:
 142                                 temp = cos(x) + sin(x);
  58                                 temp =  cos(x)+sin(x);
 143                                 break;
 144                         case 1:
 145                                 temp = -cos(x) + sin(x);
  61                                 temp = -cos(x)+sin(x);
 146                                 break;
 147                         case 2:
 148                                 temp = -cos(x) - sin(x);
  64                                 temp = -cos(x)-sin(x);
 149                                 break;
 150                         case 3:
 151                                 temp = cos(x) - sin(x);
  67                                 temp =  cos(x)-sin(x);
 152                                 break;
 153                         }

 155                         b = invsqrtpi * temp / sqrt(x);
  70                         b = invsqrtpi*temp/sqrt(x);
 156                 } else {
 157                         a = j0(x);
 158                         b = j1(x);

 160 #endif /* ! codereview */
 161                         for (i = 1; i < n; i++) {
 162                                 temp = b;
 163                                 /* avoid underflow */
 164                                 b = b * ((GENERIC)(i + i) / x) - a;
  74                                 b = b*((GENERIC)(i+i)/x) - a;
 165                                 a = temp;
 166                         }
 167                 }
 168         } else {
 169                 if (x < 1e-9) {         /* use J(n,x) = 1/n!*(x/2)^n */
 170                         b = pow(0.5 * x, (GENERIC)n);

  80                         b = pow(0.5*x, (GENERIC) n);
 172                         if (b != zero) {
 173                                 for (a = one, i = 1; i <= n; i++)
 174                                         a *= (GENERIC)i;

 176                                 b = b / a;
  84                                 b = b/a;
 177                         }
 178                 } else {
```



```
 179                         /*
 180                          * use backward recurrence
 181                          *                      x         x^2     x^2
 182                          *  J(n,x)/J(n-1,x) =  ----   ------   ------   .....
 183                          *                      2n  - 2(n+1) - 2(n+2)
 184                          *
 185                          *                      1         1         1
 186                          *  (for large x)   =  ----  ------   ------   .....
 187                          *                      2n   2(n+1)   2(n+2)
 188                          *                      -- - ------ - ------ -
 189                          *                       x       x              x
 190                          *
 191                          * Let w = 2n/x and h = 2/x, then the above quotient
 192                          * is equal to the continued fraction:
 193                          *                 1
 194                          *      = ----------------------
 195                          *                       1
 196                          *         w - -----------------
 197                          *                      1
 198                          *                    w+h - ---------
 199                          *                       w+2h - ...
 200                          *
 201                          * To determine how many terms needed, let
 202                          * Q(0) = w, Q(1) = w(w+h) - 1,
 203                          * Q(k) = (w+k*h)*Q(k-1) - Q(k-2),
 204                          * When Q(k) > 1e4      good for single
 205                          * When Q(k) > 1e9      good for double
 206                          * When Q(k) > 1e17     good for quaduple
 207                          */
 208                         /* determine k */
 209                         GENERIC t, v;
 210                         double q0, q1, h, tmp;
 211                         int k, m;

 213                         w = (n + n) / (double)x;
 214                         h = 2.0 / (double)x;
 120                         w  = (n+n)/(double)x;
 121                         h = 2.0/(double)x;
 215                         q0 = w;
 216                         z = w + h;
 217                         q1 = w * z - 1.0;
 124                         q1 = w*z - 1.0;
 218                         k = 1;

 220                         while (q1 < 1.0e9) {
 221                                 k += 1;
 222                                 z += h;
 223                                 tmp = z * q1 - q0;
 130                                 tmp = z*q1 - q0;
 224                                 q0 = q1;
 225                                 q1 = tmp;
 226                         }

 228                         m = n + n;

 230                         for (t = zero, i = 2 * (n + k); i >= m; i -= 2)
 231                                 t = one / (i / x - t);

 134                         m = n+n;
 135                         for (t = zero, i = 2*(n+k); i >= m; i -= 2)
 136                                 t = one/(i/x-t);
 233                         a = t;
 234                         b = one;

 236                         /* BEGIN CSTYLED */
 237 #endif /* ! codereview */
```

```
238                         /*
239                          * estimate log((2/x)^n*n!) = n*log(2/x)+n*ln(n)
240                          * hence, if n*(log(2n/x)) > ...
241                          *  single:
242                          *    8.8722839355e+01
243                          *  double:
244                          *    7.09782712893383973096e+02
245                          *  long double:
246                          *    1.13565234062941439494919310779707650060 70e+04
247                          * then recurrent value may overflow and the result is
248                          * likely underflow to zero
249                          */
250                         /* END CSTYLED */
251 #endif /* ! codereview */
252                         tmp = n;
253                         v = two / x;
254                         tmp = tmp * log(fabs(v * tmp));

139                         v = two/x;
140                         tmp = tmp*log(fabs(v*tmp));
256                         if (tmp < 7.09782712893383973096e+02) {
257                                 for (i = n - 1; i > 0; i--) {
142                                 for (i = n-1; i > 0; i--) {
258                                         temp = b;
259                                         b = ((i + i) / x) * b - a;
144                                         b = ((i+i)/x)*b - a;
260                                         a = temp;
261                                 }
262                         } else {
263                                 for (i = n - 1; i > 0; i--) {
148                                 for (i = n-1; i > 0; i--) {
264                                         temp = b;
265                                         b = ((i + i) / x) * b - a;
150                                         b = ((i+i)/x)*b - a;
266                                         a = temp;

268 #endif /* ! codereview */
269                                         if (b > 1e100) {
270                                                 a /= b;
271                                                 t /= b;
272                                                 b = 1.0;
273                                         }
274                                 }
275                         }

277                         b = (t * j0(x) / b);
152                         b = (t*j0(x)/b);
278                 }
279         }

281 #endif /* ! codereview */
282         if (sgn != 0)
283                 return (-b);
284         else
285                 return (b);
286 }

288 GENERIC
289 yn(int n, GENERIC x)
290 {
291         int i;
292         int sign;
293         GENERIC a, b, temp = 0, ox, on;

295         ox = x;
296         on = (GENERIC)n;
```



```
298 #endif /* ! codereview */
299         if (isnan(x))
300                 return (x * x);         /* + -> * for Cheetah */

155                 return (x*x);   /* + -> * for Cheetah */
302         if (x <= zero) {
303                 if (x == zero) {
304                         /* return -one/zero; */
305                         return (_SVID_libm_err((GENERIC)n, x, 12));
306                 } else {
307                         /* return zero/zero; */
308                         return (_SVID_libm_err((GENERIC)n, x, 13));
309                 }
310         }

312         if (!((int)_lib_version == libm_ieee || (__xpg6 &
313             _C99SUSv3_math_errexcept) != 0)) {
165         if (!((int)_lib_version == libm_ieee ||
166             (__xpg6 & _C99SUSv3_math_errexcept) != 0)) {
314                 if (x > X_TLOSS)
315                         return (_SVID_libm_err(on, ox, 39));
316         }

318 #endif /* ! codereview */
319         sign = 1;

321 #endif /* ! codereview */
322         if (n < 0) {
323                 n = -n;

325                 if ((n & 1) == 1)
326                         sign = -1;
170                 if ((n&1) == 1) sign = -1;
327         }

329 #endif /* ! codereview */
330         if (n == 0)
331                 return (y0(x));

333 #endif /* ! codereview */
334         if (n == 1)
335                 return (sign * y1(x));

172                 return (sign*y1(x));
337         if (!finite(x))
338                 return (zero);

340         if (x > 1.0e91) {
341                 /*
342                  * x >> n**2
343                  *  Jn(x) = cos(x-(2n+1)*pi/4)*sqrt(2/x*pi)
344                  *  Yn(x) = sin(x-(2n+1)*pi/4)*sqrt(2/x*pi)
345                  *  Let s = sin(x), c = cos(x),
346                  *  xn = x-(2n+1)*pi/4, sqt2 = sqrt(2), then
347                  *
348                  *    n sin(xn)*sqt2    cos(xn)*sqt2
349                  *      ----------------------------------
350                  *       0       s-c             c+s
351                  *       1      -s-c            -c+s
352                  *       2      -s+c            -c-s
353                  *       3       s+c             c-s
354                  */
355                 switch (n & 3) {
191                 switch (n&3) {
356                 case 0:
```

```
357                         temp = sin(x) - cos(x);
193                         temp =  sin(x)-cos(x);
358                         break;
359                 case 1:
360                         temp = -sin(x) - cos(x);
196                         temp = -sin(x)-cos(x);
361                         break;
362                 case 2:
363                         temp = -sin(x) + cos(x);
199                         temp = -sin(x)+cos(x);
364                         break;
365                 case 3:
366                         temp = sin(x) + cos(x);
202                         temp =  sin(x)+cos(x);
367                         break;
368                 }

370                 b = invsqrtpi * temp / sqrt(x);
205                 b = invsqrtpi*temp/sqrt(x);
371         } else {
372                 a = y0(x);
373                 b = y1(x);

375 #endif /* ! codereview */
376                 /*
377                  * fix 1262058 and take care of non-default rounding
378                  */
379                 for (i = 1; i < n; i++) {
380                         temp = b;
381                         b *= (GENERIC)(i + i) / x;

209                         b *= (GENERIC) (i + i) / x;
383                         if (b <= -DBL_MAX)
384                                 break;

386 #endif /* ! codereview */
387                         b -= a;
388                         a = temp;
389                 }
390         }

392 #endif /* ! codereview */
393         if (sign > 0)
394                 return (b);
395         else
396                 return (-b);
397 }
```

```
**********************************************************
    1338 Thu Jun  6 20:37:52 2019
new/usr/src/lib/libm/common/C/lgamma.c
11210 libm should be cstyle(10NBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

#endif /* ! codereview */
/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2005 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __lgamma = lgamma

#include "libm.h"

extern int signgam;

double
lgamma(double x)
{
lgamma(double x) {
        double g;

        if (!finite(x))
                return (x * x);

        g = rint(x);

#endif /* ! codereview */
        if (x == g && x <= 0.0) {
                signgam = 1;
                return (_SVID_libm_err(x, x, 15));
        }

        g = __k_lgamma(x, &signgam);

#endif /* ! codereview */
        if (!finite(g))
                g = _SVID_libm_err(x, x, 14);
```



```
#endif /* ! codereview */
        return (g);
}
```

```
**********************************************************
    1338 Thu Jun  6 20:37:52 2019
new/usr/src/lib/libm/common/C/lgamma_r.c
11210 libm should be cstyle(10NBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

#endif /* ! codereview */
/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2005 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __lgamma_r = lgamma_r

#include "libm.h"

double
lgamma_r(double x, int *signgamp)
{
lgamma_r(double x, int *signgamp) {
        double g;

        if (isnan(x))
                return (x * x);

        g = rint(x);

#endif /* ! codereview */
        if (x == g && x <= 0.0) {
                *signgamp = 1;
                return (_SVID_libm_err(x, x, 15));
        }

        g = __k_lgamma(x, signgamp);

#endif /* ! codereview */
        if (!finite(g))
                g = _SVID_libm_err(x, x, 14);

#endif /* ! codereview */
        return (g);
```



```
}
```

```
**********************************************************
    5133 Thu Jun  6 20:37:53 2019
new/usr/src/lib/libm/common/C/libm.h
11210 libm should be cstyle(10NBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */
/*
 * Copyright 2005 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#ifndef _LIBM_H
#define         _LIBM_H

#include <sys/isa_defs.h>

#ifdef _ASM
/* BEGIN CSTYLED */

/*
 * Disable amd64 assembly code profiling for now.
 */
#if defined(__amd64)
#undef PROF
#endif

#include <sys/asm_linkage.h>

#define         NAME(x) x
#define         TEXT    .section        ".text"
#define         DATA    .section        ".data"
#define         RO_DATA .section        ".rodata"
#define         IDENT(x)        .ident  x

#if defined(__sparc)

#define         LIBM_ANSI_PRAGMA_WEAK(sym,stype) \
        .weak __/**/sym;                 \
        .type __/**/sym,#stype;          \
__/**/sym       = sym

#ifndef SET_FILE
#define         SET_FILE(x) \
        .file   x
```



```
#endif  /* !defined(SET_FILE) */

#ifdef PIC
/*
 * One should *never* pass o7 to PIC_SETUP.
 */
#define         PIC_SETUP(via) \
9:      call    8f; \
        sethi   %hi(NAME(_GLOBAL_OFFSET_TABLE_)-(9b-.)),%via; \
8:      or      %via,%lo(NAME(_GLOBAL_OFFSET_TABLE_)-(9b-.)),%via; \
        add     %via,%o7,%via
/*
 * Must save/restore %o7 in leaf routines; may *not* use jmpl!
 */
#define         PIC_LEAF_SETUP(via) \
        or      %g0,%o7,%g1; \
9:      call    8f; \
        sethi   %hi(NAME(_GLOBAL_OFFSET_TABLE_)-(9b-.)),%via; \
8:      or      %via,%lo(NAME(_GLOBAL_OFFSET_TABLE_)-(9b-.)),%via; \
        add     %via,%o7,%via; \
        or      %g0,%g1,%o7
#ifdef __sparcv9
#define         PIC_SET(via,sym,dst)    ldx     [%via+sym],%dst
#else   /* defined(__sparcv9) */
#define         PIC_SET(via,sym,dst)    ld      [%via+sym],%dst
#endif  /* defined(__sparcv9) */
#else   /* defined(PIC) */
#define         PIC_SETUP(via)
#define         PIC_LEAF_SETUP(via)
#ifdef __sparcv9
/*
 * g1 is used as scratch register in V9 mode
 */
#define         PIC_SET(via,sym,dst)    setx    sym,%g1,%dst
#else   /* defined(__sparcv9) */
#define         PIC_SET(via,sym,dst)    set     sym,%dst
#endif  /* defined(__sparcv9) */
#endif  /* defined(PIC) */

/*
 * Workaround for 4337025: MCOUNT in asm_linkage.h does not support __sparcv9
 */
#if defined(PROF) && defined(__sparcv9)

#undef MCOUNT_SIZE
#undef MCOUNT

#if !defined(PIC)
#define         MCOUNT_SIZE     (9*4)   /* 9 instructions */
#define         MCOUNT(x) \
        save    %sp, -SA(MINFRAME), %sp; \
        sethi   %hh(.L_/**/x/**/1), %o0; \
        sethi   %lm(.L_/**/x/**/1), %o1; \
        or      %o0, %hm(.L_/**/x/**/1), %o0; \
        or      %o1, %lo(.L_/**/x/**/1), %o1; \
        sllx    %o0, 32, %o0; \
        call    _mcount; \
        or      %o0, %o1, %o0; \
        restore; \
        .common .L_/**/x/**/1, 8, 8
#elif defined(PIC32)
#define         MCOUNT_SIZE     (10*4)  /* 10 instructions */
#define         MCOUNT(x) \
        save    %sp,-SA(MINFRAME),%sp; \
1:      call    .+8; \
        sethi   %hi(_GLOBAL_OFFSET_TABLE_-(1b-.)),%o0; \
```

```
        sethi   %hi(.L_/**/x/**/1),%o1; \
        add     %o0,%lo(_GLOBAL_OFFSET_TABLE_-(1b-.)),%o0; \
        add     %o1,%lo(.L_/**/x/**/1),%o1; \
        add     %o0,%o7,%o0; \
        call    _mcount; \
        ldx     [%o0+%o1],%o0; \
        restore; \
        .common .L_/**/x/**/1,8,8
#else   /* PIC13 */
#define         MCOUNT_SIZE     (8*4)   /* 8 instructions */
#define         MCOUNT(x) \
        save    %sp,-SA(MINFRAME),%sp; \
1:      call    .+8; \
        sethi   %hi(_GLOBAL_OFFSET_TABLE_-(1b-.)),%o0; \
        add     %o0,%lo(_GLOBAL_OFFSET_TABLE_-(1b-.)),%o0; \
        add     %o0,%o7,%o0; \
        call    _mcount; \
        ldx     [%o0+%lo(.L_/**/x/**/1)],%o0; \
        restore; \
        .common .L_/**/x/**/1,8,8
#endif  /* !defined(PIC) */
#endif /* defined(PROF) && defined(__sparcv9) */

#elif defined(__x86)

#define         LIBM_ANSI_PRAGMA_WEAK(sym,stype) \
        .weak __/**/sym;                \
        .type __/**/sym,@stype;         \
__/**/sym       = sym

#ifdef PIC
#if defined(__amd64)
#define         PIC_SETUP(x)
#define         PIC_WRAPUP
#define         PIC_F(x)        x@PLT
#define         PIC_G(x)        x@GOTPCREL(%rip)
#define         PIC_L(x)        x(%rip)
#define         PIC_G_LOAD(insn,sym,dst) \
        movq    PIC_G(sym),%dst; \
        insn    (%dst),%dst
#else
#define         PIC_SETUP(label) \
        pushl   %ebx; \
        call    .label; \
.label: popl    %ebx; \
        addl    $_GLOBAL_OFFSET_TABLE_+[.-.label],%ebx
#define         PIC_WRAPUP      popl    %ebx
#define         PIC_F(x)        x@PLT
#define         PIC_G(x)        x@GOT(%ebx)
#define         PIC_L(x)        x@GOTOFF(%ebx)
#define         PIC_G_LOAD(insn,sym,dst) \
        mov     PIC_G(sym),%dst; \
        insn    (%dst),%dst
#endif
#else   /* defined(PIC) */
#define         PIC_SETUP(x)
#define         PIC_WRAPUP
#define         PIC_F(x)        x
#define         PIC_G(x)        x
#define         PIC_L(x)        x
#define         PIC_G_LOAD(insn,sym,dst)        insn    sym,%dst
#endif  /* defined(PIC) */

#else
#error Unknown architecture
#endif
```



```
/* END CSTYLED */
#else   /* defined(_ASM) */

#include "libm_macros.h"
#include "libm_protos.h"
#include "libm_inlines.h"
#include <math.h>
#if defined(__SUNPRO_C)
#include <sunmath.h>
#endif

#endif  /* defined(_ASM) */

#endif  /* _LIBM_H */
```

```
**********************************************************
    2404 Thu Jun  6 20:37:53 2019
new/usr/src/lib/libm/common/C/libm_macros.h
11210 libm should be cstyle(10NBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

#endif /* ! codereview */
/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2005 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#ifndef _LIBM_MACROS_H
#define _LIBM_MACROS_H

#include <sys/isa_defs.h>

#if defined(__sparc)

#define HIWORD          0
#define LOWORD          1
#define HIXWORD         0               /* index of int containing exponent */
#define XSGNMSK         0x80000000      /* exponent bit mask within the int */
#define XBIASED_EXP(x)  ((((int *)&x)[HIXWORD] & ~0x80000000) >> 16)
#define ISZEROL(x)      (((((int *)&x)[0] & ~XSGNMSK) | ((int *)&x)[1] | \
        ((int *)&x)[2] | ((int *)&x)[3]) == 0)

#elif defined(__x86)

#define HIWORD          1
#define LOWORD          0
#define HIXWORD         2
#define XSGNMSK         0x8000
#define XBIASED_EXP(x)  (((int *)&x)[HIXWORD] & 0x7fff)
#define ISZEROL(x)      (x == 0.0L)

#define HANDLE_UNSUPPORTED

/*
 * "convert" the high-order 32 bits of a SPARC quad precision
 * value ("I") to the sign, exponent, and high-order bits of an
```



```
 * x86 extended double precision value ("E"); the low-order bits
 * in the 12-byte quantity are left intact
 */
#define ITOX(I, E) \
        E[2] = 0xffff & ((I) >> 16); \
        E[1] = (((I) & 0x7fff0000) == 0) ? (E[1] & 0x7fff) | (0x7fff8000 & \
        ((I) << 15)) : 0x80000000 | (E[1] & 0x7fff) | (0x7fff8000 & ((I) << \
        15))
                E[1] = (((I) & 0x7fff0000) == 0)? \
                    (E[1] & 0x7fff) | (0x7fff8000 & ((I) << 15)) :\
                    0x80000000 | (E[1] & 0x7fff) | (0x7fff8000 & ((I) << 15))

/*
 * "convert" the sign, exponent, and high-order bits of an x86
 * extended double precision value ("E") to the high-order 32 bits
 * of a SPARC quad precision value ("I")
 */
#define XTOI(E, I) \
        I = ((E[2] << 16) | (0xffff & (E[1] >> 15)))
                I = ((E[2]<<16) | (0xffff & (E[1]>>15)))

#else
#error Unknown architecture
#endif

#endif /* _LIBM_MACROS_H */
```

```
**********************************************************
    5464 Thu Jun  6 20:37:54 2019
new/usr/src/lib/libm/common/C/libm_protos.h
11210 libm should be cstyle(10NBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

#endif /* ! codereview */
/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2005 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#ifndef _C_LIBM_PROTOS_H
#define _C_LIBM_PROTOS_H

/*
 * Many symbols used to be namespaced with __libm to prevent collisions.  All
 * but these two were otherwise scoped local and directly bound, so that
 * collision could not occur.
 *
 * For reasons unknown, these two are global (but private).
 */
#define __rem_pio2              __libm__rem_pio2
#define __rem_pio2m             __libm__rem_pio2m

#ifndef _ASM
#ifdef __STDC__
#define __P(p)                  p
#else
#define __P(p)                  ()
#endif

#include <sys/ieeefp.h>

extern double _SVID_libm_err __P((double, double, int));
extern double __k_cos __P((double, double));
extern double __k_cos_ __P((double *));
extern double __k_lgamma __P((double, int *));
extern double __k_sin __P((double, double));
extern double __k_sin_ __P((double *));
extern double __k_sincos __P((double, double, double *));
```



```
extern double __k_sincos_ __P((double *, double *));
extern double __k_tan __P((double, double, int));
extern double __k_cexp __P((double, int *));
extern long double __k_cexpl __P((long double, int *));
extern double __k_clog_r __P((double, double, double *));
extern long double __k_clog_rl __P((long double, long double, long double *));
extern double __k_atan2 __P((double, double, double *));
extern long double __k_atan2l __P((long double, long double, long double *));
extern int __rem_pio2 __P((double, double *));
extern int __rem_pio2m __P((double *, double *, int, int, int, const int *));

/*
 * entry points that are in-lined
 */
extern double copysign __P((double, double));
extern int finite __P((double));
extern enum fp_class_type fp_class __P((double));
extern double infinity __P((void));
extern int isinf __P((double));
extern int signbit __P((double));

/*
 * new C99 entry points
 */
extern double fdim __P((double, double));
extern double fma __P((double, double, double));
extern double fmax __P((double, double));
extern double fmin __P((double, double));
extern double frexp __P((double, int *));
extern double ldexp __P((double, int));
extern double modf __P((double, double *));
extern double nan __P((const char *));
extern double nearbyint __P((double));
extern double nexttoward __P((double, long double));
extern double remquo __P((double, double, int *));
extern double round __P((double));
extern double scalbln __P((double, long int));
extern double tgamma __P((double));
extern double trunc __P((double));
extern float fdimf __P((float, float));
extern float fmaf __P((float, float, float));
extern float fmaxf __P((float, float));
extern float fminf __P((float, float));
extern float frexpf __P((float, int *));
extern float ldexpf __P((float, int));
extern float modff __P((float, float *));
extern float nanf __P((const char *));
extern float nearbyintf __P((float));
extern float nextafterf __P((float, float));
extern float nexttowardf __P((float, long double));
extern float remquof __P((float, float, int *));
extern float roundf __P((float));
extern float scalblnf __P((float, long int));
extern float tgammaf __P((float));
extern float truncf __P((float));
extern long double frexpl(long double, int *);
extern long double fdiml __P((long double, long double));
extern long double fmal __P((long double, long double, long double));
extern long double fmaxl __P((long double, long double));
extern long double fminl __P((long double, long double));
extern long double ldexpl __P((long double, int));
extern long double modfl __P((long double, long double *));
extern long double nanl __P((const char *));
extern long double nearbyintl __P((long double));
extern long double nextafterl __P((long double, long double));
extern long double nexttowardl __P((long double, long double));
```

```
extern long double remquol __P((long double, long double, int *));
extern long double roundl __P((long double));
extern long double scalblnl __P((long double, long int));
extern long double tgammal __P((long double));
extern long double truncl __P((long double));
extern long int lrint __P((double));
extern long int lrintf __P((float));
extern long int lrintl __P((long double));
extern long int lround __P((double));
extern long int lroundf __P((float));
extern long int lroundl __P((long double));
extern long long int llrint __P((double));
extern long long int llrintf __P((float));
extern long long int llrintl __P((long double));
extern long long int llround __P((double));
extern long long int llroundf __P((float));
extern long long int llroundl __P((long double));
#endif /* _ASM */

#endif /* _C_LIBM_PROTOS_H */
```

```
**********************************************************
    7925 Thu Jun  6 20:37:54 2019
new/usr/src/lib/libm/common/C/log.c
11210 libm should be cstyle(10NBLD) clean
**********************************************************
   1 /*
   2  * CDDL HEADER START
   3  *
   4  * The contents of this file are subject to the terms of the
   5  * Common Development and Distribution License (the "License").
   6  * You may not use this file except in compliance with the License.
   7  *
   8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
   9  * or http://www.opensolaris.org/os/licensing.
  10  * See the License for the specific language governing permissions
  11  * and limitations under the License.
  12  *
  13  * When distributing Covered Code, include this CDDL HEADER in each
  14  * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  15  * If applicable, add the following below this CDDL HEADER, with the
  16  * fields enclosed by brackets "[]" replaced with your own identifying
  17  * information: Portions Copyright [yyyy] [name of copyright owner]
  18  *
  19  * CDDL HEADER END
  20  */

  22 #endif /* ! codereview */
  23 /*
  24  * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
  25  */

  27 #endif /* ! codereview */
  28 /*
  29  * Copyright 2005 Sun Microsystems, Inc.  All rights reserved.
  30  * Use is subject to license terms.
  31  */

  33 #pragma weak __log = log


  21 /* INDENT OFF */
  36 /*
  37  * log(x)
  38  * Table look-up algorithm with product polynomial approximation.
  39  * By K.C. Ng, Oct 23, 2004. Updated Oct 18, 2005.
  40  *
  41  * (a). For x in [1-0.125, 1+0.1328125], using a special approximation:
  42  *      Let f = x - 1 and z = f*f.
  43  *      return f + ((a1*z) *
  44  *                  ((a2 + (a3*f)*(a4+f)) + (f*z)*(a5+f))) *
  45  *                  (((a6 + f*(a7+f)) + (f*z)*(a8+f)) *
  46  *                  ((a9 + (a10*f)*(a11+f)) + (f*z)*(a12+f)))
  47  * a1   -6.88821452420390473170286327331268694251775741577e-0002,
  48  * a2    1.97493380704769294631262255279580131173133850098e+0000,
  49  * a3    2.24963218866067560242072431719861924648284912109e+0000,
  50  * a4   -9.02975906958474405783476868236903101205825805664e-0001,
  51  * a5   -1.47391630715542865104339398385491222143173217773e+0000,
  52  * a6    1.86846544648220058704168877738993614912033081055e+0000,
  53  * a7    1.82277370459347465292410106485476717352867126465e+0000,
  54  * a8    1.25295479915214102994980294170090928673744201660e+0000,
  55  * a9    1.96709676945198275177517643896862864494323730469e+0000,
  56  * a10  -4.00127989749189894030934055990655906498432159424e-0001,
  57  * a11   3.01675528558798333733648178167641162872314453125e+0000,
  58  * a12  -9.52325445049240770778453679668018594384193420410e-0001,
  59  *
  60  *      with remez error |(log(1+f) - P(f))/f| <= 2**-56.81 and
```



```
  61  *
  62  * (b). For 0.09375 <= x < 24
  63  *      Use an 8-bit table look-up (3-bit for exponent and 5 bit for
  64  *      significand):
  65  *      Let ix stands for the high part of x in IEEE double format.
  66  *      Since 0.09375 <= x < 24, we have
  67  *                  0x3fb80000 <= ix < 0x40380000.
  68  *      Let j = (ix - 0x3fb80000) >> 15. Then  0 <= j < 256. Choose
  69  *      a Y[j] such that HIWORD(Y[j]) ~ 0x3fb8400 + (j<<15) (the middle
  70  *      number between 0x3fb80000 + (j<<15) and 3fb80000 + ((j+1)<<15)),
  71  *      and at the same time 1/Y[j] as well as log(Y[j]) are very close
  72  *      to 53-bits floating point numbers.
  73  *      A table of Y[j], 1/Y[j], and log(Y[j]) are pre-computed and thus
  74  *              log(x)  = log(Y[j]) + log(1 + (x-Y[j])*(1/Y[j]))
  75  *                      = log(Y[j]) + log(1 + s)
  76  *      where
  77  *              s = (x-Y[j])*(1/Y[j])
  78  *      We compute max (x-Y[j])*(1/Y[j]) for the chosen Y[j] and obtain
  79  *      |s| < 0.0154. By applying remez algorithm with Product Polynomial
  80  *      Approximiation, we find the following approximated of log(1+s)
  81  *              (b1*s)*(b2+s*(b3+s))*((b4+s*b5)+(s*s)*(b6+s))*(b7+s*(b8+s))
  82  *      with remez error |log(1+s) - P(s)| <= 2**-63.5
  83  *
  84  * (c). Otherwise, get "n", the exponent of x, and then normalize x to
  85  *      z in [1,2). Then similar to (b) find a Y[i] that matches z to 5.5
  86  *      significant bits. Then
  87  *          log(x) = n*ln2 + log(Y[i]) + log(z/Y[i]).
  88  *
  89  * Special cases:
  90  *      log(x) is NaN with signal if x < 0 (including -INF) ;
  91  *      log(+INF) is +INF; log(0) is -INF with signal;
  92  *      log(NaN) is that NaN with no signal.
  93  *
  94  * Maximum error observed: less than 0.90 ulp
  95  *
  96  * Constants:
  97  * The hexadecimal values are the intended ones for the following constants.
  98  * The decimal values may be used, provided that the compiler will convert
  99  * from decimal to binary accurately enough to produce the hexadecimal values
 100  * shown.
 101  */
  88 /* INDENT ON */

 103 #include "libm.h"

 105 extern const double _TBL_log[];

 107 static const double P[] = {
 108 /* ONE   */
 109         1.0,
  95 /* ONE   */  1.0,
 110 /* TWO52 */ 4503599627370496.0,
 111 /* LN2HI */ 6.93147180369123816490e-01, /* 3fe62e42, fee00000 */
 112 /* LN2LO */ 1.90821492927058770002e-10, /* 3dea39ef, 35793c76 */
 113 /* A1    */ -6.88821452420390473170286327331268694251775741577e-0002,
 114 /* A2    */ 1.97493380704769294631262255279580131173133850098e+0000,
 115 /* A3    */ 2.24963218866067560242072431719861924648284912109e+0000,
 116 /* A4    */ -9.02975906958474405783476868236903101205825805664e-0001,
 117 /* A5    */ -1.47391630715542865104339398385491222143173217773e+0000,
 118 /* A6    */ 1.86846544648220058704168877738993614912033081055e+0000,
 119 /* A7    */ 1.82277370459347465292410106485476717352867126465e+0000,
 120 /* A8    */ 1.25295479915214102994980294170090928673744201660e+0000,
 121 /* A9    */ 1.96709676945198275177517643896862864494323730469e+0000,
 122 /* A10   */ -4.00127989749189894030934055990655906498432159424e-0001,
 123 /* A11   */ 3.01675528558798333733648178167641162872314453125e+0000,
 124 /* A12   */ -9.52325445049240770778453679668018594384193420410e-0001,
```

```
125 /* B1    */ -1.250416415892836585754821498994715511798858642 58e-0001,
126 /* B2    */ 1.871617132833551518913811279146427253376131234 82e+0000,
127 /* B3    */ -1.890829562957315079785303169046528637409210205 08e+0000,
128 /* B4    */ -2.505628916736402533871341802296228706836700439 45e+0000,
129 /* B5    */ 1.648228280852583660376353691390249878168106079 10e+0000,
130 /* B6    */ -1.244091070658683406691125128418207168579101562 50e+0000,
131 /* B7    */ 1.705342316582204142960677017981652170419692993 16e+0000,
132 /* B8    */ 1.991968337846556469372671926976181566715240478 52e+0000,
133 };

135 #define ONE             P[0]
136 #define TWO52           P[1]
137 #define LN2HI           P[2]
138 #define LN2LO           P[3]
139 #define A1              P[4]
140 #define A2              P[5]
141 #define A3              P[6]
142 #define A4              P[7]
143 #define A5              P[8]
144 #define A6              P[9]
145 #define A7              P[10]
146 #define A8              P[11]
147 #define A9              P[12]
148 #define A10             P[13]
149 #define A11             P[14]
150 #define A12             P[15]
151 #define B1              P[16]
152 #define B2              P[17]
153 #define B3              P[18]
154 #define B4              P[19]
155 #define B5              P[20]
156 #define B6              P[21]
157 #define B7              P[22]
158 #define B8              P[23]

160 double
161 log(double x)
162 {
147 log(double x) {
163         double *tb, dn, dn1, s, z, r, w;
164         int i, hx, ix, n, lx;

166         n = 0;
167         hx = ((int *)&x)[HIWORD];
168         ix = hx & 0x7fffffff;
169         lx = ((int *)&x)[LOWORD];

171         /* subnormal,0,negative,inf,nan */
172         if ((hx + 0x100000) < 0x200000) {
173                 if (ix > 0x7ff00000 || (ix == 0x7ff00000 && lx != 0)) /* nan */
174                         return (x * x);

176 #endif /* ! codereview */
177                 if (((hx << 1) | lx) == 0) /* zero */
178                         return (_SVID_libm_err(x, x, 16));

180 #endif /* ! codereview */
181                 if (hx < 0)     /* negative */
182                         return (_SVID_libm_err(x, x, 17));

184 #endif /* ! codereview */
185                 if (((hx - 0x7ff00000) | lx) == 0) /* +inf */
186                         return (x);

188                 /* x must be positive and subnormal */
189                 x *= TWO52;
```



```
190                 n = -52;
191                 ix = ((int *)&x)[HIWORD];
192                 lx = ((int *)&x)[LOWORD];
193         }

195         i = ix >> 19;

197 #endif /* ! codereview */
198         if (i >= 0x7f7 && i <= 0x806) {
199                 /* 0.09375 (0x3fb80000) <= x < 24 (0x40380000) */
200                 if (ix >= 0x3fec0000 && ix < 0x3ff22000) {
201                         /* 0.875 <= x < 1.125 */
202                         s = x - ONE;
203                         z = s * s;

205 #endif /* ! codereview */
206                         if (((ix - 0x3ff00000) | lx) == 0)      /* x = 1 */
207                                 return (z);

209 #endif /* ! codereview */
210                         r = (A10 * s) * (A11 + s);
211                         w = z * s;
212                         return (s + ((A1 * z) * (A2 + ((A3 * s) * (A4 + s) + w *
213                             (A5 + s)))) * ((A6 + (s * (A7 + s) + w *
214                             (A8 + s))) * (A9 + (r + w * (A12 + s)))));
160                         return (s + ((A1 * z) *
161                                 (A2 + ((A3 * s) * (A4 + s) + w * (A5 + s)))) *
162                                 ((A6 + (s * (A7 + s) + w * (A8 + s))) *
163                                 (A9 + (r + w * (A12 + s)))));
215                 } else {
216                         i = (ix - 0x3fb80000) >> 15;
217                         tb = (double *)_TBL_log + (i + i + i);
218                         s = (x - tb[0]) * tb[1];
219                         return (tb[2] + ((B1 * s) * (B2 + s * (B3 + s))) *
220                             (((B4 + s * B5) + (s * s) * (B6 + s)) *
221                             (B7 + s * (B8 + s))));
222                 }
223         } else {
224                 dn = (double)(n + ((ix >> 20) - 0x3ff));
225                 dn1 = dn * LN2HI;
226                 i = (ix & 0x000fffff) | 0x3ff00000;     /* scale x to [1,2] */
227                 ((int *)&x)[HIWORD] = i;
228                 i = (i - 0x3fb80000) >> 15;
229                 tb = (double *)_TBL_log + (i + i + i);
230                 s = (x - tb[0]) * tb[1];
231                 dn = dn * LN2LO + tb[2];
232                 return (dn1 + (dn + ((B1 * s) * (B2 + s * (B3 + s))) *
233                     (((B4 + s * B5) + (s * s) * (B6 + s)) *
234                     (B7 + s * (B8 + s)))));
235         }
236 }
```

_____*unchanged_portion_omitted_*

```
**********************************************************
    7168 Thu Jun  6 20:37:55 2019
new/usr/src/lib/libm/common/C/log10.c
11210 libm should be cstyle(10NBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

#endif /* ! codereview */
/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2005 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __log10 = log10


/* INDENT OFF */
/*
 * log10(x) = log(x)/log10
 *
 * Base on Table look-up algorithm with product polynomial
 * approximation for log(x).
 *
 * By K.C. Ng, Nov 29, 2004
 *
 * (a). For x in [1-0.125, 1+0.125], from log.c we have
 *      log(x) =  f + ((a1*f^2) *
 *                  ((a2 + (a3*f)*(a4+f)) + (f^3)*(a5+f))) *
 *                  (((a6 + f*(a7+f)) + (f^3)*(a8+f)) *
 *                  ((a9 + (a10*f)*(a11+f)) + (f^3)*(a12+f)))
 *      where f = x - 1.
 *      (i) modify a1 <- a1 / log10
 *      (ii) 1/log10 = 0.4342944819...
 *                     = 0.4375 - 0.003205518... (7 bit shift)
 *           Let lgv = 0.4375 - 1/log10, then
 *           lgv = 0.00320551809674817234887108108 3395...,
 *      (iii) f*0.4375 is exact because f has 3 trailing zero.
 *      (iv) Thus, log10(x) = f*0.4375 - (lgv*f - PPoly)
 *
 * (b). For 0.09375 <= x < 24
 *      Let j = (ix - 0x3fb80000) >> 15. Look up Y[j], 1/Y[j], and log(Y[j])
 *      from _TBL_log.c. Then
```



```
 *              log10(x)  = log10(Y[j]) + log10(1 + (x-Y[j])*(1/Y[j]))
 *                        = log(Y[j])(1/log10) + log10(1 + s)
 *      where
 *              s = (x-Y[j])*(1/Y[j])
 *      From log.c, we have log(1+s) =
 *                                  2                2                    2
 *              (b s) (b + b s + s ) [b + b s + s (b + s)] (b + b s + s )
 *                1     2   3          4   5        6        7   8
 *
 *      By setting b1 <- b1/log10, we have
 *              log10(x) = 0.4375 * T - (lgv * T - POLY(s))
 *
 * (c). Otherwise, get "n", the exponent of x, and then normalize x to
 *      z in [1,2). Then similar to (b) find a Y[i] that matches z to 5.5
 *      significant bits. Then
 *          log(x) = n*ln2 + log(Y[i]) + log(z/Y[i]).
 *          log10(x) = n*(ln2/ln10) + log10(z).
 *
 * Special cases:
 *      log10(x) is NaN with signal if x < 0 (including -INF) ;
 *      log10(+INF) is +INF; log10(0) is -INF with signal;
 *      log10(NaN) is that NaN with no signal.
 *
 * Maximum error observed: less than 0.89 ulp
 *
 * Constants:
 * The hexadecimal values are the intended ones for the following constants.
 * The decimal values may be used, provided that the compiler will convert
 * from decimal to binary accurately enough to produce the hexadecimal values
 * shown.
 */
/* INDENT ON */

#include "libm.h"

extern const double _TBL_log[];

static const double P[] = {
/* ONE   */
        1.0,
/* ONE   */  1.0,
/* TWO52 */ 4503599627370496.0,
/* LNAHI */ 3.01029995607677847147e-01, /* 3FD34413 50900000 */
/* LNALO */ 5.63033480667509769841e-11, /* 3DCEF3FD E623E256 */
/* A1    */ -2.91425219601365825073854807070445828021 84e-02,
/* A2    */ 1.99628461483039965074226529395673424005508422852e+0000,
/* A3    */ 2.26812367662950720159642514772713184356689453125e+0000,
/* A4    */ -9.05030639084976384900471657601883634924888610840e-0001,
/* A5    */ -1.48275767132434044270894446526654064655303955078e+0000,
/* A6    */ 1.88158320939722756293122074566781520843505859375e+0000,
/* A7    */ 1.83309386046986411145098827546462416648864746094e+0000,
/* A8    */ 1.24847063988317086291601754055591300129890441895e+0000,
/* A9    */ 1.98372421445537705508854742220137268304824829102e+0000,
/* A10   */ -3.94711735767898475035764249696512706577777862549e-0001,
/* A11   */ 3.07890395362954372160402272129431366920471191406e+0000,
/* A12   */ -9.60099585275022149311041630426188930869102478027e-0001,
/* B1    */ -5.43048949503500529608380967524915402866 89e-02,
/* B2    */ 1.87161713283355151891381127914642725337613123482e+0000,
/* B3    */ -1.89082956295731507978530316904652863740921020508e+0000,
/* B4    */ -2.50562891673640253387134180229622870683670043945e+0000,
/* B5    */ 1.64822828085258366037635369139024987816810607910e+0000,
/* B6    */ -1.24409107065868340669112512841820716857910156250e+0000,
/* B7    */ 1.70534231658220414296067701798165217041969299316e+0000,
/* B8    */ 1.99196833784655646937267192697618156671524047852e+0000,
/* LGH   */ 0.4375,
/* LGL   */ 0.00320551809674817234887108108 3395,
```

```
/* LG10V */ 0.43429448190325182765112891891660509576226,
};

#define ONE             P[0]
#define TWO52           P[1]
#define LNAHI           P[2]
#define LNALO           P[3]
#define A1              P[4]
#define A2              P[5]
#define A3              P[6]
#define A4              P[7]
#define A5              P[8]
#define A6              P[9]
#define A7              P[10]
#define A8              P[11]
#define A9              P[12]
#define A10             P[13]
#define A11             P[14]
#define A12             P[15]
#define B1              P[16]
#define B2              P[17]
#define B3              P[18]
#define B4              P[19]
#define B5              P[20]
#define B6              P[21]
#define B7              P[22]
#define B8              P[23]
#define LGH             P[24]
#define LGL             P[25]
#define LG10V           P[26]

double
log10(double x)
{
log10(double x) {
        double *tb, dn, dn1, s, z, r, w;
        int i, hx, ix, n, lx;

        n = 0;
        hx = ((int *)&x)[HIWORD];
        ix = hx & 0x7fffffff;
        lx = ((int *)&x)[LOWORD];

        /* subnormal,0,negative,inf,nan */
        if ((hx + 0x100000) < 0x200000) {
                if (ix > 0x7ff00000 || (ix == 0x7ff00000 && lx != 0)) /* nan */
                        return (x * x);

#endif /* ! codereview */
                if (((hx << 1) | lx) == 0) /* zero */
                        return (_SVID_libm_err(x, x, 18));

#endif /* ! codereview */
                if (hx < 0)     /* negative */
                        return (_SVID_libm_err(x, x, 19));

#endif /* ! codereview */
                if (((hx - 0x7ff00000) | lx) == 0) /* +inf */
                        return (x);

                /* x must be positive and subnormal */
                x *= TWO52;
                n = -52;
                ix = ((int *)&x)[HIWORD];
                lx = ((int *)&x)[LOWORD];
        }
```



```
        i = ix >> 19;

#endif /* ! codereview */
        if (i >= 0x7f7 && i <= 0x806) {
                /* 0.09375 (0x3fb80000) <= x < 24 (0x40380000) */
                if (ix >= 0x3fec0000 && ix < 0x3ff20000) {
                        /* 0.875 <= x < 1.125 */
                        s = x - ONE;
                        z = s * s;

#endif /* ! codereview */
                        if (((ix - 0x3ff00000) | lx) == 0)      /* x = 1 */
                                return (z);

#endif /* ! codereview */
                        r = (A10 * s) * (A11 + s);
                        w = z * s;
                        return (LGH * s - (LGL * s - ((A1 * z) *
                            ((A2 + (A3 * s) * (A4 + s)) + w * (A5 + s))) *
                            (((A6 + s * (A7 + s)) + w * (A8 + s)) *
                            ((A9 + r) + w * (A12 + s)))));
                } else {
                        i = (ix - 0x3fb80000) >> 15;
                        tb = (double *)_TBL_log + (i + i + i);
                        s = (x - tb[0]) * tb[1];
                        return (LGH * tb[2] - (LGL * tb[2] - ((B1 * s) *
                            (B2 + s * (B3 + s))) * (((B4 + s * B5) + (s * s) *
                            (B6 + s)) * (B7 + s * (B8 + s)))));
                                (B2 + s * (B3 + s))) *
                                (((B4 + s * B5) + (s * s) * (B6 + s)) *
                                (B7 + s * (B8 + s)))));
                }
        } else {
                dn = (double)(n + ((ix >> 20) - 0x3ff));
                dn1 = dn * LNAHI;
                i = (ix & 0x000fffff) | 0x3ff00000;     /* scale x to [1,2] */
                ((int *)&x)[HIWORD] = i;
                i = (i - 0x3fb80000) >> 15;
                tb = (double *)_TBL_log + (i + i + i);
                s = (x - tb[0]) * tb[1];
                dn = dn * LNALO + tb[2] * LG10V;
                return (dn1 + (dn + ((B1 * s) * (B2 + s * (B3 + s))) *
                return (dn1 + (dn + ((B1 * s) *
                        (B2 + s * (B3 + s))) *
                    (((B4 + s * B5) + (s * s) * (B6 + s)) *
                    (B7 + s * (B8 + s)))));
        }
}
_____unchanged_portion_omitted_
```

```
**********************************************************
    6390 Thu Jun  6 20:37:55 2019
new/usr/src/lib/libm/common/C/log1p.c
11210 libm should be cstyle(10NBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

#endif /* ! codereview */
/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2005 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __log1p = log1p


/* INDENT OFF */
/*
 * Method :
 *   1. Argument Reduction: find k and f such that
 *                        1+x = 2^k * (1+f),
 *         where  sqrt(2)/2 < 1+f < sqrt(2) .
 *
 *      Note. If k=0, then f=x is exact. However, if k != 0, then f
 *      may not be representable exactly. In that case, a correction
 *      term is need. Let u=1+x rounded. Let c = (1+x)-u, then
 *      log(1+x) - log(u) ~ c/u. Thus, we proceed to compute log(u),
 *      and add back the correction term c/u.
 *      (Note: when x > 2**53, one can simply return log(x))
 *
 *   2. Approximation of log1p(f).
 *      Let s = f/(2+f) ; based on log(1+f) = log(1+s) - log(1-s)
 *               = 2s + 2/3 s**3 + 2/5 s**5 + .....,
 *               = 2s + s*R
 *      We use a special Reme algorithm on [0,0.1716] to generate
 *      a polynomial of degree 14 to approximate R The maximum error
 *      of this polynomial approximation is bounded by 2**-58.45. In
 *      other words,
 *                        2      4      6      8      10      12      14
 *          R(z) ~ Lp1*s +Lp2*s +Lp3*s +Lp4*s +Lp5*s  +Lp6*s  +Lp7*s
 *      (the values of Lp1 to Lp7 are listed in the program)
 *      and
```



```
 *          |      2          14          |     -58.45
 *          | Lp1*s +...+Lp7*s    -  R(z) | <= 2
 *          |                             |
 *      Note that 2s = f - s*f = f - hfsq + s*hfsq, where hfsq = f*f/2.
 *      In order to guarantee error in log below 1ulp, we compute log
 *      by
 *              log1p(f) = f - (hfsq - s*(hfsq+R)).
 *
 *      3. Finally, log1p(x) = k*ln2 + log1p(f).
 *                           = k*ln2_hi+(f-(hfsq-(s*(hfsq+R)+k*ln2_lo)))
 *         Here ln2 is splitted into two floating point number:
 *                        ln2_hi + ln2_lo,
 *         where n*ln2_hi is always exact for |n| < 2000.
 *
 * Special cases:
 *      log1p(x) is NaN with signal if x < -1 (including -INF) ;
 *      log1p(+INF) is +INF; log1p(-1) is -INF with signal;
 *      log1p(NaN) is that NaN with no signal.
 *
 * Accuracy:
 *      according to an error analysis, the error is always less than
 *      1 ulp (unit in the last place).
 *
 * Constants:
 * The hexadecimal values are the intended ones for the following
 * constants. The decimal values may be used, provided that the
 * compiler will convert from decimal to binary accurately enough
 * to produce the hexadecimal values shown.
 *
 * Note: Assuming log() return accurate answer, the following
 *       algorithm can be used to compute log1p(x) to within a few ULP:
 *
 *              u = 1+x;
 *              if (u == 1.0) return x ; else
 *                          return log(u)*(x/(u-1.0));
 *
 *       See HP-15C Advanced Functions Handbook, p.193.
 */
/* INDENT ON */

#include "libm.h"

static const double xxx[] = {
/* ln2_hi */
        6.931471803691238164 90e-01,            /* 3fe62e42 fee00000 */
/* ln2_hi */    6.931471803691238164 90e-01,    /* 3fe62e42 fee00000 */
/* ln2_lo */ 1.908214929270587700 02e-10,       /* 3dea39ef 35793c76 */
/* two54 */ 1.801439850948198400 00e+16,        /* 43500000 00000000 */
/* Lp1 */ 6.666666666666735130e-01,             /* 3FE55555 55555593 */
/* Lp2 */ 3.999999999940941908e-01,             /* 3FD99999 9997FA04 */
/* Lp3 */ 2.857142874366239149e-01,             /* 3FD24924 94229359 */
/* Lp4 */ 2.222219843214978396e-01,             /* 3FCC71C5 1D8E78AF */
/* Lp5 */ 1.818357216161805012e-01,             /* 3FC74664 96CB03DE */
/* Lp6 */ 1.531383769920937332e-01,             /* 3FC39A09 D078C69F */
/* Lp7 */ 1.479819860511658591e-01,             /* 3FC2F112 DF3E5244 */
/* zero */ 0.0
};

#endif /* ! codereview */
#define ln2_hi          xxx[0]
#define ln2_lo          xxx[1]
#define two54           xxx[2]
#define Lp1             xxx[3]
#define Lp2             xxx[4]
#define Lp3             xxx[5]
#define Lp4             xxx[6]
```

```
#define Lp5             xxx[7]
#define Lp6             xxx[8]
#define Lp7             xxx[9]
#define zero            xxx[10]

double
log1p(double x)
{
log1p(double x) {
        double hfsq, f, c = 0.0, s, z, R, u;
        int k, hx, hu, ax;

        hx = ((int *)&x)[HIWORD]; /* high word of x */
        ax = hx & 0x7fffffff;

        if (ax >= 0x7ff00000) { /* x is inf or nan */
                if (((hx - 0xfff00000) | ((int *)&x)[LOWORD]) == 0) /* -inf */
                        return (_SVID_libm_err(x, x, 44));

#endif /* ! codereview */
                return (x * x);
        }

        k = 1;

#endif /* ! codereview */
        if (hx < 0x3FDA827A) {                  /* x < 0.41422  */
                if (ax >= 0x3ff00000)           /* x <= -1.0 */
                        return (_SVID_libm_err(x, x, x == -1.0 ? 43 : 44));

#endif /* ! codereview */
                if (ax < 0x3e200000) {          /* |x| < 2**-29 */
                        if (two54 + x > zero && /* raise inexact */
                            ax < 0x3c900000)    /* |x| < 2**-54 */
                                return (x);
                        else
                                return (x - x * x * 0.5);
                }

#endif /* ! codereview */
                if (hx > 0 || hx <= (int)0xbfd2bec3) {  /* -0.2929<x<0.41422 */
                        k = 0;
                        f = x;
                        hu = 1;
                }
        }

#endif /* ! codereview */
        /* We will initialize 'c' here. */
        if (k != 0) {
                if (hx < 0x43400000) {
                        u = 1.0 + x;
                        hu = ((int *)&u)[HIWORD];       /* high word of u */
                        k = (hu >> 20) - 1023;

#endif /* ! codereview */
                        /*
                         * correction term
                         */
                        c = k > 0 ? 1.0 - (u - x) : x - (u - 1.0);
                        c /= u;
                } else {
                        u = x;
                        hu = ((int *)&u)[HIWORD];       /* high word of u */
                        k = (hu >> 20) - 1023;
                        c = 0;
```



```
                }

#endif /* ! codereview */
                hu &= 0x000fffff;

#endif /* ! codereview */
                if (hu < 0x6a09e) {     /* normalize u */
                        ((int *)&u)[HIWORD] = hu | 0x3ff00000;
                } else {                /* normalize u/2 */
                        k += 1;
                        ((int *)&u)[HIWORD] = hu | 0x3fe00000;
                        hu = (0x00100000 - hu) >> 2;
                }

#endif /* ! codereview */
                f = u - 1.0;
        }

#endif /* ! codereview */
        hfsq = 0.5 * f * f;

#endif /* ! codereview */
        if (hu == 0) {                  /* |f| < 2**-20 */
                if (f == zero) {
                        if (k == 0)
                                return (zero);

#endif /* ! codereview */
                        /* We already initialized 'c' before, when (k != 0) */
                        c += k * ln2_lo;
                        return (k * ln2_hi + c);
                }

#endif /* ! codereview */
                R = hfsq * (1.0 - 0.66666666666666666 * f);

#endif /* ! codereview */
                if (k == 0)
                        return (f - R);

#endif /* ! codereview */
                return (k * ln2_hi - ((R - (k * ln2_lo + c)) - f));
        }

#endif /* ! codereview */
        s = f / (2.0 + f);
        z = s * s;
        R = z * (Lp1 + z * (Lp2 + z * (Lp3 + z * (Lp4 + z * (Lp5 + z * (Lp6 +
            z * Lp7))))));

        R = z * (Lp1 + z * (Lp2 + z * (Lp3 + z * (Lp4 + z * (Lp5 +
                z * (Lp6 + z * Lp7))))));
        if (k == 0)
                return (f - (hfsq - s * (hfsq + R)));

        return (k * ln2_hi -
            ((hfsq - (s * (hfsq + R) + (k * ln2_lo + c))) - f));
        return (k * ln2_hi - ((hfsq - (s * (hfsq + R) +
                (k * ln2_lo + c))) - f));
}
```

_____*unchanged_portion_omitted_*

```
**********************************************************
    7326 Thu Jun  6 20:37:56 2019
new/usr/src/lib/libm/common/C/log2.c
11210 libm should be cstyle(10NBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __log2 = log2


/* INDENT OFF */
/*
 * log2(x) = log(x)/log2
 *
 * Base on Table look-up algorithm with product polynomial
 * approximation for log(x).
 *
 * By K.C. Ng, Nov 29, 2004
 *
 * (a). For x in [1-0.125, 1+0.125], from log.c we have
 *      log(x) =  f + ((a1*f^2) *
 *                 ((a2 + (a3*f)*(a4+f)) + (f^3)*(a5+f))) *
 *                 (((a6 + f*(a7+f)) + (f^3)*(a8+f)) *
 *                 ((a9 + (a10*f)*(a11+f)) + (f^3)*(a12+f)))
 *      where f = x - 1.
 *      (i) modify a1 <- a1 / log2
 *      (ii) 1/log2 = 1.4426950408889634...
 *                  = 1.5 - 0.057304959... (4 bit shift)
 *           Let lv = 1.5 - 1/log2, then
 *           lv = 0.05730495911103659264007531899810795666 5325,
 *      (iii) f*1.5 is exact because f has 3 trailing zero.
 *      (iv) Thus, log2(x) = f*1.5 - (lv*f - PPoly)
 *
 * (b). For 0.09375 <= x < 24
 *      Let j = (ix - 0x3fb80000) >> 15. Look up Y[j], 1/Y[j], and log(Y[j])
 *      from _TBL_log.c. Then
 *              log2(x)  = log2(Y[j]) + log2(1 + (x-Y[j])*(1/Y[j]))
```



```
 *                       = log(Y[j])(1/log2) + log2(1 + s)
 *      where
 *              s = (x-Y[j])*(1/Y[j])
 *      From log.c, we have log(1+s) =
 *                                 2              2                    2
 *              (b s) (b + b s + s ) [b + b s + s (b + s)] (b + b s + s )
 *                1     2   3          4   5        6        7   8
 *
 *      By setting b1 <- b1/log2, we have
 *              log2(x) = 1.5 * T - (lv * T - POLY(s))
 *
 * (c). Otherwise, get "n", the exponent of x, and then normalize x to
 *      z in [1,2). Then similar to (b) find a Y[i] that matches z to 5.5
 *      significant bits. Then
 *          log2(x) = n + log2(z).
 *
 * Special cases:
 *      log2(x) is NaN with signal if x < 0 (including -INF) ;
 *      log2(+INF) is +INF; log2(0) is -INF with signal;
 *      log2(NaN) is that NaN with no signal.
 *
 * Maximum error observed: less than 0.84 ulp
 *
 * Constants:
 * The hexadecimal values are the intended ones for the following constants.
 * The decimal values may be used, provided that the compiler will convert
 * from decimal to binary accurately enough to produce the hexadecimal values
 * shown.
 */
/* INDENT ON */

#include "libm.h"
#include "libm_protos.h"

extern const double _TBL_log[];

static const double P[] = {
/* ONE   */
        1.0,
/* ONE   */  1.0,
/* TWO52 */ 4503599627370496.0,
/* LN10V */ 1.4426950408889634073599246810018920433347, /* 1/log10 */
/* ZERO  */ 0.0,
/* A1    */ -9.6809362455249638217841932228967194640116e-02,
/* A2    */ 1.9962846148303996507422652939567342400550842 2852e+0000,
/* A3    */ 2.2681236766295072015964251477271318435668945 3125e+0000,
/* A4    */ -9.0503063908497638490047165760188363492488861 0840e-0001,
/* A5    */ -1.4827576713243404427089444652665406465530395 5078e+0000,
/* A6    */ 1.8815832093972275629312207456678152084350585 9375e+0000,
/* A7    */ 1.8330938604698641114509882754646241664886474 6094e+0000,
/* A8    */ 1.2484706398831708629160175405559130012989044 1895e+0000,
/* A9    */ 1.9837242144553770550885474222013726830482482 9102e+0000,
/* A10   */ -3.9471173576789847503576424969651270657777786 2549e-0001,
/* A11   */ 3.0789039536295437216040227212943136692047119 1406e+0000,
/* A12   */ -9.6009958527502214931104163042618893086910247 8027e-0001,
/* B1    */ -1.8039695622547469514898963204616532885451e-01,
/* B2    */ 1.8716171328335515189138112791464272533761312 3482e+0000,
/* B3    */ -1.8908295629573150797853031690465286374092102 0508e+0000,
/* B4    */ -2.5056289167364025338713418022962287068367004 3945e+0000,
/* B5    */ 1.6482282808525836603763536913902498781681060 7910e+0000,
/* B6    */ -1.2440910706586834066911251284182071685791015 6250e+0000,
/* B7    */ 1.7053423165822041429606770179816521704196929 9316e+0000,
/* B8    */ 1.9919683378465564693726719269761815667152404 7852e+0000,
/* LGH   */ 1.5,
/* LGL   */ 0.05730495911103659264007531899810795666 5325,
};
```

```
126 #define ONE             P[0]
127 #define TWO52           P[1]
128 #define LN10V           P[2]
129 #define ZERO            P[3]
130 #define A1              P[4]
131 #define A2              P[5]
132 #define A3              P[6]
133 #define A4              P[7]
134 #define A5              P[8]
135 #define A6              P[9]
136 #define A7              P[10]
137 #define A8              P[11]
138 #define A9              P[12]
139 #define A10             P[13]
140 #define A11             P[14]
141 #define A12             P[15]
142 #define B1              P[16]
143 #define B2              P[17]
144 #define B3              P[18]
145 #define B4              P[19]
146 #define B5              P[20]
147 #define B6              P[21]
148 #define B7              P[22]
149 #define B8              P[23]
150 #define LGH             P[24]
151 #define LGL             P[25]

153 double
154 log2(double x)
155 {
145 log2(double x) {
156         int i, hx, ix, n, lx;

158         n = 0;
159         hx = ((int *)&x)[HIWORD];
160         ix = hx & 0x7fffffff;
161         lx = ((int *)&x)[LOWORD];
149         hx = ((int *) &x)[HIWORD]; ix = hx & 0x7fffffff;
150         lx = ((int *) &x)[LOWORD];

163         /* subnormal,0,negative,inf,nan */
164         if ((hx + 0x100000) < 0x200000) {
165 #if defined(FPADD_TRAPS_INCOMPLETE_ON_NAN)
166                 if (ix >= 0x7ff80000) /* assumes sparc-like QNaN */
167                         return (x);   /* for Cheetah when x is QNaN */
168 #endif

170 #endif /* ! codereview */
171                 if (((hx << 1) | lx) == 0)              /* log(0.0) = -inf */
172                         return (A5 / fabs(x));

174 #endif /* ! codereview */
175                 if (hx < 0) {                           /* x < 0 */
176                         if (ix >= 0x7ff00000)
177                                 return (x - x);         /* x is -inf or NaN */
178                         else
179                                 return (ZERO / (x - x));
180                 }

182 #endif /* ! codereview */
183                 if (((hx - 0x7ff00000) | lx) == 0)      /* log(inf) = inf */
184                         return (x);

186 #endif /* ! codereview */
187                 if (ix >= 0x7ff00000)                   /* log(NaN) = NaN */
```



```
188                         return (x - x);

190 #endif /* ! codereview */
191                 x *= TWO52;
192                 n = -52;
193                 hx = ((int *)&x)[HIWORD];
194                 ix = hx & 0x7fffffff;
195                 lx = ((int *)&x)[LOWORD];
158                 hx = ((int *) &x)[HIWORD]; ix = hx & 0x7fffffff;
159                 lx = ((int *) &x)[LOWORD];
196         }

198         /* 0.09375 (0x3fb80000) <= x < 24 (0x40380000) */
199         i = ix >> 19;

201 #endif /* ! codereview */
202         if (i >= 0x7f7 && i <= 0x806) {
203                 /* 0.875 <= x < 1.125 */
204                 if (ix >= 0x3fec0000 && ix < 0x3ff20000) {
205                         double s, z, r, w;

207                         s = x - ONE;
208                         z = s * s;
209                         r = (A10 * s) * (A11 + s);
164                         s = x - ONE; z = s * s; r = (A10 * s) * (A11 + s);
210                         w = z * s;

212                         if (((ix << 12) | lx) == 0) {
166                         if (((ix << 12) | lx) == 0)
213                                 return (z);
214                         } else {
215                                 return (LGH * s - (LGL * s - ((A1 * z) * ((A2 +
216                                     (A3 * s) * (A4 + s)) + w * (A5 + s))) *
168                         else
169                                 return (LGH * s - (LGL * s - ((A1 * z) *
170                                 ((A2 + (A3 * s) * (A4 + s)) + w * (A5 + s))) *
217                                     (((A6 + s * (A7 + s)) + w * (A8 + s)) *
218                                     ((A9 + r) + w * (A12 + s)))));
219                         }
220 #endif /* ! codereview */
221                 } else {
222                         double *tb, s;

224 #endif /* ! codereview */
225                         i = (ix - 0x3fb80000) >> 15;
226                         tb = (double *)_TBL_log + (i + i + i);

173                         tb = (double *) _TBL_log + (i + i + i);
228                         if (((ix << 12) | lx) == 0)     /* 2's power */
229                                 return ((double)((ix >> 20) - 0x3ff));

175                                 return ((double) ((ix >> 20) - 0x3ff));
231                         s = (x - tb[0]) * tb[1];
232                         return (LGH * tb[2] - (LGL * tb[2] - ((B1 * s) *
233                             (B2 + s * (B3 + s))) * (((B4 + s * B5) + (s * s) *
234                             (B6 + s)) * (B7 + s * (B8 + s)))));
178                                 (B2 + s * (B3 + s))) *
179                                 (((B4 + s * B5) + (s * s) * (B6 + s)) *
180                                 (B7 + s * (B8 + s)))));
235                 }
236         } else {
237                 double *tb, dn, s;

239                 dn = (double)(n + ((ix >> 20) - 0x3ff));
184                 dn = (double) (n + ((ix >> 20) - 0x3ff));
240                 ix <<= 12;
```

```
#endif /* ! codereview */
                if ((ix | lx) == 0)
                        return (dn);

                i = ((unsigned)ix >> 12) | 0x3ff00000;  /* scale x to [1,2) */
                ((int *)&x)[HIWORD] = i;
                i = ((unsigned) ix >> 12) | 0x3ff00000; /* scale x to [1,2) */
                ((int *) &x)[HIWORD] = i;
                i = (i - 0x3fb80000) >> 15;
                tb = (double *)_TBL_log + (i + i + i);
                tb = (double *) _TBL_log + (i + i + i);
                s = (x - tb[0]) * tb[1];
                return (dn + (tb[2] * LN10V + ((B1 * s) * (B2 + s * (B3 + s))) *
                return (dn + (tb[2] * LN10V + ((B1 * s) *
                        (B2 + s * (B3 + s))) *
                    (((B4 + s * B5) + (s * s) * (B6 + s)) *
                    (B7 + s * (B8 + s)))));
        }
}
```

_____*unchanged_portion_omitted_*

```
**********************************************************
    2141 Thu Jun  6 20:37:56 2019
new/usr/src/lib/libm/common/C/logb.c
11210 libm should be cstyle(10NBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __logb = logb
#pragma weak _logb = logb

#include "libm.h"
#include "xpg6.h"                       /* __xpg6 */
#define _C99SUSv3_logb  _C99SUSv3_logb_subnormal_is_like_ilogb

#if defined(__x86)
static const double two52 = 4503599627370496.0;
#else
/*
 * v: high part of a non-zero subnormal |x|; w: low part of |x|
 */
static int
ilogb_subnormal(unsigned v, unsigned w)
{
ilogb_subnormal(unsigned v, unsigned w) {
        int r = -1022 - 52;

        if (v)
                r += 32;
        else
                v = w;

#endif /* ! codereview */
        if (v & 0xffff0000)
                r += 16, v >>= 16;

#endif /* ! codereview */
        if (v & 0xff00)
```



```
                r += 8, v >>= 8;

#endif /* ! codereview */
        if (v & 0xf0)
                r += 4, v >>= 4;

#endif /* ! codereview */
        v <<= 1;
        return (r + ((0xffffaa50 >> v) & 0x3));
}
#endif /* defined(__x86) */

double
logb(double x)
{
        int *px = (int *)&x, k = px[HIWORD] & ~0x80000000;
logb(double x) {
        int *px = (int *) &x, k = px[HIWORD] & ~0x80000000;

        if (k < 0x00100000) {
                if ((px[LOWORD] | k) == 0) {
                if ((px[LOWORD] | k) == 0)
                        return (_SVID_libm_err(x, x, 45));
                } else if ((__xpg6 & _C99SUSv3_logb) != 0) {
                else if ((__xpg6 & _C99SUSv3_logb) != 0) {
#if defined(__x86)
                        x *= two52;
                        return ((double)(((px[HIWORD] & 0x7ff00000) >> 20) -
                            1075));
                        return ((double) (((px[HIWORD] & 0x7ff00000) >> 20)
                                - 1075));
#else
                        return ((double)ilogb_subnormal(k, px[LOWORD]));
                        return ((double) ilogb_subnormal(k, px[LOWORD]));
#endif
                } else {
                } else
                        return (-1022.0);
                }
        } else if (k < 0x7ff00000) {
                return ((double)((k >> 20) - 1023));
        } else {
        } else if (k < 0x7ff00000)
                return ((double) ((k >> 20) - 1023));
        else
                return (x * x);
        }
#endif /* ! codereview */
}
```

```
**********************************************************
    1119 Thu Jun  6 20:37:57 2019
new/usr/src/lib/libm/common/C/matherr.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak matherr = __matherr

#include "libm.h"

/* ARGSUSED0 */
int
__matherr(struct exception *x)
{
__matherr(struct exception *x) {
        return (0);
}
______unchanged_portion_omitted_
```

```
**********************************************************
    2289 Thu Jun  6 20:37:57 2019
new/usr/src/lib/libm/common/C/nextafter.c
11210 libm should be cstyle(10NBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

#endif /* ! codereview */
/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2005 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __nextafter = nextafter
#pragma weak _nextafter = nextafter

#include "libm.h"
#include <float.h>                      /* DBL_MIN */

double
nextafter(double x, double y)
{
nextafter(double x, double y) {
        int hx, hy, k;
        double ans;
        unsigned lx;
        volatile double dummy __unused;

        hx = ((int *)&x)[HIWORD];
        lx = ((int *)&x)[LOWORD];
        hy = ((int *)&y)[HIWORD];
        k = (hx & ~0x80000000) | lx;

        if (x == y)
                return (y);             /* C99 requirement */

#endif /* ! codereview */
        if (x != x || y != y)
                return (x * y);

#endif /* ! codereview */
        if (k == 0) {                   /* x = 0 */
```



```
                k = hy & 0x80000000;
                ((int *)&ans)[HIWORD] = k;
                ((int *)&ans)[LOWORD] = 1;
        } else if (hx >= 0) {
                if (x > y) {
                        ((int *)&ans)[LOWORD] = lx - 1;
                        k = (lx == 0) ? hx - 1 : hx;
                        k = (lx == 0)? hx - 1 : hx;
                        ((int *)&ans)[HIWORD] = k;
                } else {
                        ((int *)&ans)[LOWORD] = lx + 1;
                        k = (lx == 0xffffffff) ? hx + 1 : hx;
                        k  = (lx == 0xffffffff)? hx + 1 : hx;
                        ((int *)&ans)[HIWORD] = k;
                }
        } else {
                if (x < y) {
                        ((int *)&ans)[LOWORD] = lx - 1;
                        k = (lx == 0) ? hx - 1 : hx;
                        k = (lx == 0)? hx - 1 : hx;
                        ((int *)&ans)[HIWORD] = k;
                } else {
                        ((int *)&ans)[LOWORD] = lx + 1;
                        k = (lx == 0xffffffff) ? hx + 1 : hx;
                        k  = (lx == 0xffffffff)? hx + 1 : hx;
                        ((int *)&ans)[HIWORD] = k;
                }
        }

#endif /* ! codereview */
        k = (k >> 20) & 0x7ff;

#endif /* ! codereview */
        if (k == 0x7ff) {
                /* overflow */
                return (_SVID_libm_err(x, y, 46));
#if !defined(__lint)
        } else if (k == 0) {
                /* underflow */
                dummy = DBL_MIN * copysign(DBL_MIN, x);
#endif
        }

#endif /* ! codereview */
        return (ans);
}
```

```
**********************************************************
   10218 Thu Jun  6 20:37:58 2019
new/usr/src/lib/libm/common/C/pow.c
11210 libm should be cstyle(10NBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __pow = pow

/*
 * pow(x,y) return x**y
 *                      n
 * Method:  Let x =  2   * (1+f)
 *      1. Compute and return log2(x) in two pieces:
 *             log2(x) = w1 + w2,
 *         where w1 has 24 bits trailing zero.
 *      2. Perform y*log2(x) by simulating muti-precision arithmetic
 *      3. Return x**y = exp2(y*log(x))
 *
 * Special cases:
 *      1.  (anything) ** +-0 is 1
 *      1'. 1 ** (anything)   is 1      (C99; 1 ** +-INF/NAN used to be NAN)
 *      2.  (anything) ** 1   is itself
 *      3.  (anything except 1) ** NAN is NAN ("except 1" is C99)
 *      4.  NAN ** (anything except 0) is NAN
 *      5.  +-(|x| > 1) **  +INF is +INF
 *      6.  +-(|x| > 1) **  -INF is +0
 *      7.  +-(|x| < 1) **  +INF is +0
 *      8.  +-(|x| < 1) **  -INF is +INF
 *      9.  -1          ** +-INF is 1   (C99; -1 ** +-INF used to be NAN)
 *      10. +0 ** (+anything except 0, NAN)              is +0
 *      11. -0 ** (+anything except 0, NAN, odd integer)  is +0
 *      12. +0 ** (-anything except 0, NAN)              is +INF
 *      13. -0 ** (-anything except 0, NAN, odd integer)  is +INF
 *      14. -0 ** (odd integer) = -( +0 ** (odd integer) )
 *      15. +INF ** (+anything except 0,NAN) is +INF
 *      16. +INF ** (-anything except 0,NAN) is +0
```



```
 *      17. -INF ** (anything)  = -0 ** (-anything)
 *      18. (-anything) ** (integer) is (-1)**(integer)*(+anything**integer)
 *      19. (-anything except 0 and inf) ** (non-integer) is NAN
 *
 * Accuracy:
 *      pow(x,y) returns x**y nearly rounded. In particular
 *                      pow(integer,integer)
 *      always returns the correct integer provided it is representable.
 */

#include "libm.h"
#include "xpg6.h"                       /* __xpg6 */
#define _C99SUSv3_pow   _C99SUSv3_pow_treats_Inf_as_an_even_int

static const double zero = 0.0,
        one = 1.0,
        two = 2.0;
static const double zero = 0.0, one = 1.0, two = 2.0;

extern const double _TBL_log2_hi[], _TBL_log2_lo[];

static const double two53 = 9007199254740992.0,
static const double
        two53 = 9007199254740992.0,
        A1_hi = 2.88539004325866699921875,
        A1_lo = 3.851925982503504196360600 2e-8,
        A1 = 2.885390081777926817222541963606002026086e+0000,
        A2 = 9.617966939207270828380543979852286255862e-0001,
        A3 = 5.770807680887875964868853124873696201995e-0001,
        B0_hi = 2.88539004325866699921875,
        B0_lo = 3.851925982253279305637432058 5e-8,
        B0 = 2.885390081777926814720293056374320585689e+0000,
        B1 = 9.617966939259755138949202350396200257632e-0001,
        B2 = 5.770780163585687000782112776448797953382e-0001,
        B3 = 4.121985488948771523290174512461778354953e-0001,
        B4 = 3.207590534812432970433641789022666850193e-0001;

static double
log2_x(double x, double *w)
{
log2_x(double x, double *w) {
        double f, s, z, qn, h, t;
        int *px = (int *)&x;
        int *pz = (int *)&z;
        int *px = (int *) &x;
        int *pz = (int *) &z;
        int i, j, ix, n;

        n = 0;
        ix = px[HIWORD];

#endif /* ! codereview */
        if (ix >= 0x3fef03f1 && ix < 0x3ff08208) {      /* 65/63 > x > 63/65 */
                double f1, v;

#endif /* ! codereview */
                f = x - one;

#endif /* ! codereview */
                if (((ix - 0x3ff00000) | px[LOWORD]) == 0) {
                        *w = zero;
                        return (zero);  /* log2(1)= +0 */
                }

#endif /* ! codereview */
                qn = one / (two + f);
```

```
122                 s = f * qn;             /* |s|<2**-6 */
123                 v = s * s;
124                 h = (double)((float)s);
125                 f1 = (double)((float)f);
 52                 h = (double) ((float) s);
 53                 f1 = (double) ((float) f);
126                 t = qn * (((f - two * h) - h * f1) - h * (f - f1));
127                 /* s = h+t */
128                 f1 = h * B0_lo + s * (v * (B1 + v * (B2 + v * (B3 + v * B4))));
129                 t = f1 + t * B0;
130                 h *= B0_hi;
131                 s = (double)((float)(h + t));
 59                 s = (double) ((float) (h + t));
132                 *w = t - (s - h);
133                 return (s);
134         }

136 #endif /* ! codereview */
137         if (ix < 0x00100000) {          /* subnormal x */
138                 x *= two53;
139                 n = -53;
140                 ix = px[HIWORD];
141         }

143 #endif /* ! codereview */
144         /* LARGE N */
145         n += ((ix + 0x1000) >> 20) - 0x3ff;
146         ix = (ix & 0x000fffff) | 0x3ff00000;    /* scale x to [1,2] */
147         px[HIWORD] = ix;
148         i = ix + 0x1000;
149         pz[HIWORD] = i & 0xffffe000;
150         pz[LOWORD] = 0;
151         qn = one / (x + z);
152         f = x - z;
153         s = f * qn;
154         h = (double)((float)s);
 63         h = (double) ((float) s);
155         t = qn * ((f - (h + h) * z) - h * f);
156         j = (i >> 13) & 0x7f;
157         f = s * s;
158         t = t * A1 + h * A1_lo;
159         t += (s * f) * (A2 + f * A3);
160         qn = h * A1_hi;
161         s = n + _TBL_log2_hi[j];
162         h = qn + s;
163         t += _TBL_log2_lo[j] - ((h - s) - qn);
164         f = (double)((float)(h + t));
 73         f = (double) ((float) (h + t));
165         *w = t - (f - h);
166         return (f);
167 }

169 extern const double _TBL_exp2_hi[], _TBL_exp2_lo[];
170 static const double             /* poly app of 2^x-1 on [-1e-10,2^-7+1e-10] */
171         E1 = 6.931471805599453100674958533810346197328e-0001,
172         E2 = 2.402265069587779347846769151717493815979e-0001,
173         E3 = 5.550410866475410512631124892773937864699e-0002,
174         E4 = 9.618143209991026824853712740162451423355e-0003,
175         E5 = 1.333357676549940345096774122231849082991e-0003;

177 double
178 pow(double x, double y)
179 {
 87 pow(double x, double y) {
180         double z, ax;
181         double y1, y2, w1, w2;
```



```
182         int sbx, sby, j, k, yisint;
183         int hx, hy, ahx, ahy;
184         unsigned lx, ly;
185         int *pz = (int *)&z;
 93         int *pz = (int *) &z;

187         hx = ((int *)&x)[HIWORD];
188         lx = ((unsigned *)&x)[LOWORD];
189         hy = ((int *)&y)[HIWORD];
190         ly = ((unsigned *)&y)[LOWORD];
 95         hx = ((int *) &x)[HIWORD];
 96         lx = ((unsigned *) &x)[LOWORD];
 97         hy = ((int *) &y)[HIWORD];
 98         ly = ((unsigned *) &y)[LOWORD];
191         ahx = hx & ~0x80000000;
192         ahy = hy & ~0x80000000;

194 #endif /* ! codereview */
195         if ((ahy | ly) == 0) {                          /* y==zero  */
196                 if ((ahx | lx) == 0)
197                         z = _SVID_libm_err(x, y, 20);   /* +-0**+-0 */
198                 else if ((ahx | (((lx | -lx) >> 31) & 1)) > 0x7ff00000)
199                         z = _SVID_libm_err(x, y, 42);   /* NaN**+-0 */
200                 else
201                         z = one;                        /* x**+-0 = 1 */

203 #endif /* ! codereview */
204                 return (z);
205         } else if (hx == 0x3ff00000 && lx == 0 && (__xpg6 & _C99SUSv3_pow) !=
206             0) {
101         } else if (hx == 0x3ff00000 && lx == 0 &&
102                 (__xpg6 & _C99SUSv3_pow) != 0)
207                 return (one);           /* C99: 1**anything = 1 */
208         } else if (ahx > 0x7ff00000 || (ahx == 0x7ff00000 && lx != 0) || ahy >
209             0x7ff00000 || (ahy == 0x7ff00000 && ly != 0)) {
104         else if (ahx > 0x7ff00000 || (ahx == 0x7ff00000 && lx != 0) ||
105                 ahy > 0x7ff00000 || (ahy == 0x7ff00000 && ly != 0))
210                 return (x * y); /* +-NaN return x*y; + -> * for Cheetah */
211         }

213 #endif /* ! codereview */
214         /* includes Sun: 1**NaN = NaN */
215         sbx = (unsigned)hx >> 31;
216         sby = (unsigned)hy >> 31;
107         sbx = (unsigned) hx >> 31;
108         sby = (unsigned) hy >> 31;
217         ax = fabs(x);

219         /*
220          * determine if y is an odd int when x < 0
221          * yisint = 0 ... y is not an integer
222          * yisint = 1 ... y is an odd int
223          * yisint = 2 ... y is an even int
224          */
225         yisint = 0;

227 #endif /* ! codereview */
228         if (sbx) {
229                 if (ahy >= 0x43400000) {
118                 if (ahy >= 0x43400000)
230                         yisint = 2;                     /* even integer y */
231                 } else if (ahy >= 0x3ff00000) {
120                 else if (ahy >= 0x3ff00000) {
232                         k = (ahy >> 20) - 0x3ff;        /* exponent */

234 #endif /* ! codereview */
```

```
                        if (k > 20) {
                                j = ly >> (52 - k);

#endif /* ! codereview */
                                if ((j << (52 - k)) == ly)
                                        yisint = 2 - (j & 1);
                        } else if (ly == 0) {
                                j = ahy >> (20 - k);

#endif /* ! codereview */
                                if ((j << (20 - k)) == ahy)
                                        yisint = 2 - (j & 1);
                        }
                }
        }

#endif /* ! codereview */
        /* special value of y */
        if (ly == 0) {
                if (ahy == 0x7ff00000) {        /* y is +-inf */
                        if (((ahx - 0x3ff00000) | lx) == 0) {
                                if ((__xpg6 & _C99SUSv3_pow) != 0)
                                        return (one);
                                /* C99: (-1)**+-inf = 1 */
                                else
                                        return (y - y);

#endif /* ! codereview */
                                /* Sun: (+-1)**+-inf = NaN */
                        } else if (ahx >= 0x3ff00000) {
                        } else if (ahx >= 0x3ff00000)
                                /* (|x|>1)**+,-inf = inf,0 */
                                return (sby == 0 ? y : zero);
                        } else {        /* (|x|<1)**-,+inf = inf,0 */
                        else                    /* (|x|<1)**-,+inf = inf,0 */
                                return (sby != 0 ? -y : zero);
                        }
                }

#endif /* ! codereview */
                if (ahy == 0x3ff00000) {        /* y is  +-1 */
                        if (sby != 0) {         /* y is -1 */
                                if (x == zero)  /* divided by zero */
                                        return (_SVID_libm_err(x, y, 23));
                                else if (ahx < 0x40000 || ((ahx - 0x40000) |
                                    lx) == 0)   /* overflow */
                                        return (_SVID_libm_err(x, y, 21));
                                else
                                        return (one / x);
                        } else {
                        } else
                                return (x);
                        }
                }

#endif /* ! codereview */
                if (hy == 0x40000000) { /* y is  2 */
                        if (ahx >= 0x5ff00000 && ahx < 0x7ff00000)
                                return (_SVID_libm_err(x, y, 21));
                        /* x*x overflow */
                        else if ((ahx < 0x1e56a09e && (ahx | lx) != 0) ||
                            (ahx == 0x1e56a09e && lx < 0x667f3bcd))
                                return (_SVID_libm_err(x, y, 22));
                        /* x*x underflow */
                        else
                                return (x * x);
```



```
                }

#endif /* ! codereview */
                if (hy == 0x3fe00000) {
                        if (!((ahx | lx) == 0 || ((ahx - 0x7ff00000) | lx) ==
                            0 || sbx == 1))
                                return (sqrt(x));       /* y is 0.5 and x > 0 */
                }
        }

#endif /* ! codereview */
        /* special value of x */
        if (lx == 0) {
                if (ahx == 0x7ff00000 || ahx == 0 || ahx == 0x3ff00000) {
                        /* x is +-0,+-inf,-1 */
                        z = ax;

#endif /* ! codereview */
                        if (sby == 1) {
                                z = one / z;    /* z = |x|**y */

#endif /* ! codereview */
                                if (ahx == 0)
                                        return (_SVID_libm_err(x, y, 23));
                        }

#endif /* ! codereview */
                        if (sbx == 1) {
                                if (ahx == 0x3ff00000 && yisint == 0)
                                        z = _SVID_libm_err(x, y, 24);
                                /* neg**non-integral is NaN + invalid */
                                else if (yisint == 1)
                                        z = -z; /* (x<0)**odd = -(|x|**odd) */
                        }

#endif /* ! codereview */
                        return (z);
                }
        }

#endif /* ! codereview */
        /* (x<0)**(non-int) is NaN */
        if (sbx == 1 && yisint == 0)
                return (_SVID_libm_err(x, y, 24));

        /*
         * Now ax is finite, y is finite
         * first compute log2(ax) = w1+w2, with 24 bits w1
         */
        /* Now ax is finite, y is finite */
        /* first compute log2(ax) = w1+w2, with 24 bits w1 */
        w1 = log2_x(ax, &w2);

        /* split up y into y1+y2 and compute (y1+y2)*(w1+w2) */
        if (((ly & 0x07fffff) == 0) || ahy >= 0x47e00000 || ahy <=
            0x38100000) {
        if (((ly & 0x07fffff) == 0) || ahy >= 0x47e00000 ||
                ahy <= 0x38100000) {
                /* no need to split if y is short or too large or too small */
                y1 = y * w1;
                y2 = y * w2;
        } else {
                y1 = (double)((float)y);
                y1 = (double) ((float) y);
                y2 = (y - y1) * w1 + y * w2;
                y1 *= w1;
```

```
        }

#endif /* ! codereview */
        z = y1 + y2;
        j = pz[HIWORD];

#endif /* ! codereview */
        if (j >= 0x40900000) {                                  /* z >= 1024 */
                if (!(j == 0x40900000 && pz[LOWORD] == 0)) {    /* z > 1024 */
                if (!(j == 0x40900000 && pz[LOWORD] == 0))      /* z > 1024 */
                        return (_SVID_libm_err(x, y, 21));      /* overflow */
                } else {
                else {
                        w2 = y1 - z;
                        w2 += y2;

#endif /* ! codereview */
                        /* rounded to inf */
                        if (w2 >= -8.0085662595372965671 60e-17)
                                return (_SVID_libm_err(x, y, 21));

#endif /* ! codereview */
                        /* overflow */
                }
        } else if ((j & ~0x80000000) >= 0x4090cc00) {           /* z <= -1075 */
                if (!(j == 0xc090cc00 && pz[LOWORD] == 0)) {    /* z < -1075 */
                if (!(j == 0xc090cc00 && pz[LOWORD] == 0))      /* z < -1075 */
                        return (_SVID_libm_err(x, y, 22));      /* underflow */
                } else {
                else {
                        w2 = y1 - z;
                        w2 += y2;

#endif /* ! codereview */
                        if (w2 <= zero) /* underflow */
                                return (_SVID_libm_err(x, y, 22));
                }
        }

#endif /* ! codereview */
        /*
         * compute 2**(k+f[j]+g)
         */
        k = (int)(z * 64.0 + (((hy ^ (ahx - 0x3ff00000)) > 0) ? 0.5 : -0.5));
        k = (int) (z * 64.0 + (((hy ^ (ahx - 0x3ff00000)) > 0) ? 0.5 : -0.5));
        j = k & 63;
        w1 = y2 - ((double)k * 0.015625 - y1);
        w1 = y2 - ((double) k * 0.015625 - y1);
        w2 = _TBL_exp2_hi[j];
        z = _TBL_exp2_lo[j] + (w2 * w1) * (E1 + w1 * (E2 + w1 * (E3 + w1 * (E4 +
            w1 * E5))));
        z = _TBL_exp2_lo[j] + (w2 * w1) * (E1 + w1 * (E2 + w1 * (E3 + w1 *
                (E4 + w1 * E5))));
        z += w2;
        k >>= 6;

#endif /* ! codereview */
        if (k < -1021)
                z = scalbn(z, k);
        else                                    /* subnormal output */
                pz[HIWORD] += k << 20;

#endif /* ! codereview */
        if (sbx == 1 && yisint == 1)
                z = -z;                         /* (-ve)**(odd int) */
```



```
#endif /* ! codereview */
        return (z);
}
```

```
**********************************************************
    2363 Thu Jun  6 20:37:58 2019
new/usr/src/lib/libm/common/C/remainder.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

#endif /* ! codereview */
/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __remainder = remainder

/*
 * remainder(x,p)
 * Code originated from 4.3bsd.
 * Modified by K.C. Ng for SUN 4.0 libm.
 * Return :
 *      returns  x REM p  =  x - [x/p]*p as if in infinite precise arithmetic,
 *      where [x/p] is the (inifinite bit) integer nearest x/p (in half way
 *      case choose the even one).
 * Method :
 *      Based on fmod() return x-[x/p]chopped*p exactly.
 */

#include "libm.h"

static const double zero = 0.0, half = 0.5;

double
remainder(double x, double p)
{
remainder(double x, double p) {
        double halfp;
        int ix, hx, hp;

        ix = ((int *)&x)[HIWORD];
        hx = ix & ~0x80000000;
        hp = ((int *)&p)[HIWORD] & ~0x80000000;
```



```
        if (hp > 0x7ff00000 || (hp == 0x7ff00000 && ((int *)&p)[LOWORD] != 0))
                return (x * p);

#endif /* ! codereview */
        if (hx > 0x7ff00000 || (hx == 0x7ff00000 && ((int *)&x)[LOWORD] != 0))
                return (x * p);

        if ((hp | ((int *)&p)[LOWORD]) == 0 || hx == 0x7ff00000)
                return (_SVID_libm_err(x, p, 28));

        p = fabs(p);

#endif /* ! codereview */
        if (hp < 0x7fe00000)
                x = fmod(x, p + p);

#endif /* ! codereview */
        x = fabs(x);

#endif /* ! codereview */
        if (hp < 0x00200000) {
                if (x + x > p) {
                        if (x == p)     /* avoid x-x=-0 in RM mode */
                                return ((ix < 0) ? -zero : zero);

                                return ((ix < 0)? -zero : zero);
                        x -= p;

#endif /* ! codereview */
                        if (x + x >= p)
                                x -= p;
                }
        } else {
                halfp = half * p;

#endif /* ! codereview */
                if (x > halfp) {
                        if (x == p)     /* avoid x-x=-0 in RM mode */
                                return ((ix < 0) ? -zero : zero);

                                return ((ix < 0)? -zero : zero);
                        x -= p;

#endif /* ! codereview */
                        if (x >= halfp)
                                x -= p;
                }
        }

        return ((ix < 0) ? -x : x);
        return ((ix < 0)? -x : x);
}
______unchanged_portion_omitted_
```

```
**********************************************************
    2051 Thu Jun  6 20:37:59 2019
new/usr/src/lib/libm/common/C/rint.c
11210 libm should be cstyle(10NBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

#endif /* ! codereview */
/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __rint = rint

/*
 * rint(x) return x rounded to integral according to the rounding direction
 * rint(x) returns result with the same sign as x's,  including 0.0.
 */

#include "libm.h"

#if defined(__i386) && !defined(__amd64) && (!defined(__FLT_EVAL_METHOD__) || \
        __FLT_EVAL_METHOD__ != 0)

#endif /* ! codereview */
extern enum fp_precision_type __swapRP(enum fp_precision_type);

#endif /* ! codereview */
#define DECLRP(x)               enum fp_precision_type x;
#define SWAPRP(new, x)          x = __swapRP(new);
#define RESTRP(x)               (void) __swapRP(x);
#else
#define DECLRP(x)
#define SWAPRP(new, x)
#define RESTRP(x)
#endif

static const double two52 = 4503599627370496.0, zero = 0.0, one = 1.0;
static const double
        two52 = 4503599627370496.0,
        zero = 0.0,
```



```
        one = 1.0;

double
rint(double x)
{
rint(double x) {
        DECLRP(rp)
        double t, w;
        int ix, hx;

        ix = ((int *)&x)[HIWORD];
        hx = ix & ~0x80000000;

        if (hx >= 0x43300000)
                return (x * one);

        t = (ix < 0) ? -two52 : two52;
        t = (ix < 0)? -two52 : two52;
        SWAPRP(fp_double, rp)           /* set precision mode to double */
        w = x + t;                      /* x+sign(x)*2**52 rounded */
        RESTRP(rp)                      /* restore precision mode */

#endif /* ! codereview */
        if (w == t)
                return ((ix < 0) ? -zero : zero);

                return ((ix < 0)? -zero : zero);
        return (w - t);
}
______unchanged_portion_omitted_
```

```
**********************************************************
    2095 Thu Jun  6 20:37:59 2019
new/usr/src/lib/libm/common/C/scalb.c
11210 libm should be cstyle(10NBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

#endif /* ! codereview */
/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2005 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __scalb = scalb
#pragma weak _scalb = scalb

#include "libm.h"

double
scalb(double x, double fn)
{
scalb(double x, double fn) {
        int hn, in, n;
        double z;

        if (isnan(x) || isnan(fn))
                return (x * fn);

        in = ((int *)&fn)[HIWORD];
        hn = in & ~0x80000000;

#endif /* ! codereview */
        if (hn == 0x7ff00000)           /* fn is inf */
                return (_SVID_libm_err(x, fn, 47));

        /* see if fn is an integer without raising inexact */
        if (hn >= 0x43300000) {
                /* |fn| >= 2^52, so it must be an integer */
                n = (in < 0) ? -65000 : 65000;
                n = (in < 0)? -65000 : 65000;
        } else if (hn < 0x3ff00000) {
                /* |fn| < 1, so it must be zero or non-integer */
```



```
                return ((fn == 0.0) ? x : (x - x) / (x - x));
                return ((fn == 0.0)? x : (x - x) / (x - x));
        } else if (hn < 0x41400000) {
                /* |fn| < 2^21 */
                if ((hn & ((1 << (0x413 - (hn >> 20))) - 1)) |
                    ((int *)&fn)[LOWORD])
                if ((hn & ((1 << (0x413 - (hn >> 20))) - 1))
                    | ((int *)&fn)[LOWORD])
                        return ((x - x) / (x - x));

#endif /* ! codereview */
                n = (int)fn;
        } else {
                if (((int *)&fn)[LOWORD] & ((1 << (0x433 - (hn >> 20))) - 1))
                        return ((x - x) / (x - x));

                n = (in < 0) ? -65000 : 65000;
                n = (in < 0)? -65000 : 65000;
        }

#endif /* ! codereview */
        z = scalbn(x, n);

#endif /* ! codereview */
        if (z != x) {
                if (z == 0.0)
                        return (_SVID_libm_err(x, fn, 33));

#endif /* ! codereview */
                if (!finite(z))
                        return (_SVID_libm_err(x, fn, 32));
        }

#endif /* ! codereview */
        return (z);
}
```

```
**********************************************************
    2782 Thu Jun  6 20:38:00 2019
new/usr/src/lib/libm/common/C/scalbn.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

#endif /* ! codereview */
/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __scalbn = scalbn

#include "libm.h"

static const double one = 1.0,
static const double
        one     = 1.0,
        huge = 1.0e300,
        tiny = 1.0e-300,
        twom54 = 5.551115123125782702118158340454101562 5e-17;

#if defined(__x86)
static const double two52 = 4503599627370496.0;
#else
/*
 * Normalize non-zero subnormal x and return biased exponent of x in [-51,0]
 */
static int
ilogb_biased(unsigned *px)
{
ilogb_biased(unsigned *px) {
        int s = 52;
        unsigned v = px[HIWORD] & ~0x80000000, w = px[LOWORD], t = v;

        if (t)
                s -= 32;
        else
                t = w;
```



```
#endif /* ! codereview */
        if (t & 0xffff0000)
                s -= 16, t >>= 16;

#endif /* ! codereview */
        if (t & 0xff00)
                s -= 8, t >>= 8;

#endif /* ! codereview */
        if (t & 0xf0)
                s -= 4, t >>= 4;

#endif /* ! codereview */
        t <<= 1;
        s -= (0xffffaa50 >> t) & 0x3;

#endif /* ! codereview */
        if (s < 32) {
                v = (v << s) | w >> (32 - s);
                w <<= s;
        } else {
                v = w << (s - 32);
                w = 0;
        }

#endif /* ! codereview */
        px[HIWORD] = (px[HIWORD] & 0x80000000) | v;
        px[LOWORD] = w;
        return (1 - s);
}
#endif /* defined(__x86) */

double
scalbn(double x, int n)
{
scalbn(double x, int n) {
        int *px, ix, hx, k;

        px = (int *)&x;
        ix = px[HIWORD];
        hx = ix & ~0x80000000;
        k = hx >> 20;

        if (k == 0x7ff)                 /* x is inf or NaN */
                return (x * one);

        if (k == 0) {
                if ((hx | px[LOWORD]) == 0 || n == 0)
                        return (x);

#endif /* ! codereview */
#if defined(__x86)
                x *= two52;
                ix = px[HIWORD];
                k = ((ix & ~0x80000000) >> 20) - 52;
#else
                k = ilogb_biased((unsigned *)px);
                ix = px[HIWORD];
#endif
                /* now k is in the range -51..0 */
                k += n;

#endif /* ! codereview */
                if (k > n)              /* integer overflow occurred */
                        k = -100;
        } else {
```

```
                /* k is in the range 1..1023 */
                k += n;

#endif /* ! codereview */
                if (k < n)              /* integer overflow occurred */
                        k = 0x7ff;
        }

        if (k > 0x7fe)
                return (huge * ((ix < 0) ? -huge : huge));

                return (huge * ((ix < 0)? -huge : huge));
        if (k < 1) {
                if (k <= -54)
                        return (tiny * ((ix < 0) ? -tiny : tiny));

                        return (tiny * ((ix < 0)? -tiny : tiny));
                k += 54;
                px[HIWORD] = (ix & ~0x7ff00000) | (k << 20);
                return (x * twom54);
        }

#endif /* ! codereview */
        px[HIWORD] = (ix & ~0x7ff00000) | (k << 20);
        return (x);
}
```

```
**********************************************************
    1068 Thu Jun  6 20:38:00 2019
new/usr/src/lib/libm/common/C/signgam.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __signgam = signgam

#include <math.h>

int signgam = 0;
```

```
**********************************************************
    1454 Thu Jun  6 20:38:01 2019
new/usr/src/lib/libm/common/C/significand.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
   1 /*
   2  * CDDL HEADER START
   3  *
   4  * The contents of this file are subject to the terms of the
   5  * Common Development and Distribution License (the "License").
   6  * You may not use this file except in compliance with the License.
   7  *
   8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
   9  * or http://www.opensolaris.org/os/licensing.
  10  * See the License for the specific language governing permissions
  11  * and limitations under the License.
  12  *
  13  * When distributing Covered Code, include this CDDL HEADER in each
  14  * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  15  * If applicable, add the following below this CDDL HEADER, with the
  16  * fields enclosed by brackets "[]" replaced with your own identifying
  17  * information: Portions Copyright [yyyy] [name of copyright owner]
  18  *
  19  * CDDL HEADER END
  20  */

  22 /*
  23  * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
  24  */

  26 #endif /* ! codereview */
  27 /*
  28  * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
  29  * Use is subject to license terms.
  30  */

  32 #pragma weak __significand = significand

  34 #include "libm.h"

  36 double
  37 significand(double x)
  38 {
  39         int ix = ((int *)&x)[HIWORD] & ~0x80000000;
  25 significand(double x) {
  26         int ix = ((int *) &x)[HIWORD] & ~0x80000000;

  41         /* weed out 0/+-Inf/NaN because C99 ilogb raises invalid on them */
  42         if ((ix | ((int *)&x)[LOWORD]) == 0 || ix >= 0x7ff00000)
  29         if ((ix | ((int *) &x)[LOWORD]) == 0 || ix >= 0x7ff00000)
  43 #if defined(FPADD_TRAPS_INCOMPLETE_ON_NAN)
  44                 return ((ix & 0x80000) != 0 ? x : x + x);

  46 #endif /* ! codereview */
  47         /* assumes sparc-like QNaN */
  48 #else
  49                 return (x + x);
  50 #endif
  51         else
  52                 return (scalbn(x, -ilogb(x)));
  53 }
```

```
**********************************************************
    5189 Thu Jun  6 20:38:01 2019
new/usr/src/lib/libm/common/C/sin.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
   1 /*
   2  * CDDL HEADER START
   3  *
   4  * The contents of this file are subject to the terms of the
   5  * Common Development and Distribution License (the "License").
   6  * You may not use this file except in compliance with the License.
   7  *
   8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
   9  * or http://www.opensolaris.org/os/licensing.
  10  * See the License for the specific language governing permissions
  11  * and limitations under the License.
  12  *
  13  * When distributing Covered Code, include this CDDL HEADER in each
  14  * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  15  * If applicable, add the following below this CDDL HEADER, with the
  16  * fields enclosed by brackets "[]" replaced with your own identifying
  17  * information: Portions Copyright [yyyy] [name of copyright owner]
  18  *
  19  * CDDL HEADER END
  20  */

  22 #endif /* ! codereview */
  23 /*
  24  * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
  25  */

  27 #endif /* ! codereview */
  28 /*
  29  * Copyright 2005 Sun Microsystems, Inc.  All rights reserved.
  30  * Use is subject to license terms.
  31  */

  33 #pragma weak __sin = sin


  21 /* INDENT OFF */
  36 /*
  37  * sin(x)
  38  * Accurate Table look-up algorithm by K.C. Ng, May, 1995.
  39  *
  40  * Algorithm: see sincos.c
  41  */

  43 #include "libm.h"

  45 static const double sc[] = {
  46 /* ONE  = */
  47         1.0,
  32 /* ONE  = */  1.0,
  48 /* NONE = */ -1.0,

  50 #endif /* ! codereview */
  51 /*
  52  * |sin(x) - (x+pp1*x^3+pp2*x^5)| <= 2^-58.79 for |x| < 0.008
  53  */
  54 /* PP1  = */-0.16666666666631655886725205237888952148062785868305556 7,
  55 /* PP2  = */.00833331565299747232356489424846675824847537497797401 7927,

  34 /* PP1  = */ -0.16666666666631655886725205237888952148062785868305556 7,
  35 /* PP2  = */   .00833331565299747232356489424846675824847537497797401 7927,
  57 /*
```



```
  58  * |(sin(x) - (x+p1*x^3+...+p4*x^9)|
  59  * |------------------------------ | <= 2^-57.63 for |x| < 0.1953125
  60  * |                x              |
  61  */
  62 /* P1   = */ -1.6666666666666296698052151389203015896 56e-0001,
  63 /* P2   = */ 8.3333333323909512956839934552803363766 63e-0003,
  64 /* P3   = */ -1.9841262379979766927915517782300984039 60e-0004,
  65 /* P4   = */ 2.7534036248542772376499876228483303511 10e-0006,

  67 #endif /* ! codereview */
  68 /*
  69  * |cos(x) - (1+qq1*x^2+qq2*x^4)| <= 2^-55.99 for |x| <= 0.008 (0x3f80624d)
  70  */
  71 /* QQ1  = */-0.49999999999754923818429119819484185427427 29,
  72 /* QQ2  = */0.04166654290435205929454520915835764039877174 0,
  45 /* QQ1  = */ -0.49999999999754923818429119819484185427427 29,
  46 /* QQ2  = */  0.04166654290435205929454520915835764039877174 0,
  73 /* PI_H = */ 3.1415926535897931159979634685,
  74 /* PI_L    = */ 1.2246467991473531772260659322 7425e-16,
  75 /* PI_L0   = */ 1.2246467991455844331128387920 5095e-16,
  76 /* PI_L1   = */ 1.7687441132271402233000052337355517376e-28,
  77 /* PI2_H   = */ 6.2831853071795862319959269370,
  78 /* PI2_L   = */ 2.4492935982947063544521318645485 0e-16,
  79 /* PI2_L0  = */ 2.4492935982911688662256775841019 0e-16,
  80 /* PI2_L1  = */ 3.5374882264542804466000104674710347 52e-28,
  81 };

  56 /* INDENT ON */

  84 #define ONEA            sc
  85 #define ONE             sc[0]
  86 #define NONE            sc[1]
  87 #define PP1             sc[2]
  88 #define PP2             sc[3]
  89 #define P1              sc[4]
  90 #define P2              sc[5]
  91 #define P3              sc[6]
  92 #define P4              sc[7]
  93 #define QQ1             sc[8]
  94 #define QQ2             sc[9]
  95 #define PI_H            sc[10]
  96 #define PI_L            sc[11]
  97 #define PI_L0           sc[12]
  98 #define PI_L1           sc[13]
  99 #define PI2_H           sc[14]
 100 #define PI2_L           sc[15]
 101 #define PI2_L0          sc[16]
 102 #define PI2_L1          sc[17]

 104 extern const double _TBL_sincos[], _TBL_sincosx[];

 106 double
 107 sin(double x)
 108 {
  81 sin(double x) {
 109         double z, y[2], w, s, v, p, q;
 110         int i, j, n, hx, ix, lx;

 112         hx = ((int *)&x)[HIWORD];
 113         lx = ((int *)&x)[LOWORD];
 114         ix = hx & ~0x80000000;

 116         if (ix <= 0x3fc50000) {         /* |x| < .1640625 */
 117                 if (ix < 0x3e400000)    /* |x| < 2**-27 */
 118                         if ((int)x == 0)
 119                                 return (x);
```

```
121 #endif /* ! codereview */
122                 z = x * x;

124 #endif /* ! codereview */
125                 if (ix < 0x3f800000)    /* |x| < 2**-8 */
126                         w = (z * x) * (PP1 + z * PP2);
127                 else
128                         w = (x * z) * ((P1 + z * P2) + (z * z) * (P3 + z * P4));

130 #endif /* ! codereview */
131                 return (x + w);
132         }

134         /* for .1640625 < x < M, */
135         n = ix >> 20;

137 #endif /* ! codereview */
138         if (n < 0x402) {                /* x < 8 */
139                 i = (((ix >> 12) & 0xff) | 0x100) >> (0x401 - n);
140                 j = i - 10;
141                 x = fabs(x);
142                 v = x - _TBL_sincosx[j];

144 #endif /* ! codereview */
145                 if (((j - 181) ^ (j - 201)) < 0) {
146                         /* near pi, sin(x) = sin(pi-x) */
147                         p = PI_H - x;
148                         i = ix - 0x400921fb;
149                         x = p + PI_L;

151 #endif /* ! codereview */
152                         if ((i | ((lx - 0x54442D00) & 0xffffff00)) == 0) {
153                                 /* very close to pi */
154                                 x = p + PI_L0;
155                                 return ((hx >= 0) ? x + PI_L1 : -(x + PI_L1));
 93                                 return ((hx >= 0)? x + PI_L1 : -(x + PI_L1));
156                         }

158 #endif /* ! codereview */
159                         z = x * x;

161 #endif /* ! codereview */
162                         if (((ix - 0x40092000) >> 11) == 0) {
163                                 /* |pi-x|<2**-8 */
164                                 w = PI_L + (z * x) * (PP1 + z * PP2);
165                         } else {
166                                 w = PI_L + (z * x) * ((P1 + z * P2) + (z * z) *
167                                     (P3 + z * P4));
 95                                 w = PI_L + (z * x) * ((P1 + z * P2) +
 96                                     (z * z) * (P3 + z * P4));
168                         }

170                         return ((hx >= 0) ? p + w : -p - w);
 98                         return ((hx >= 0)? p + w : -p - w);
171                 }

173 #endif /* ! codereview */
174                 s = v * v;

176 #endif /* ! codereview */
177                 if (((j - 382) ^ (j - 402)) < 0) {
178                         /* near 2pi, sin(x) = sin(x-2pi) */
179                         p = x - PI2_H;
180                         i = ix - 0x401921fb;
181                         x = p - PI2_L;
```



```
183 #endif /* ! codereview */
184                         if ((i | ((lx - 0x54442D00) & 0xffffff00)) == 0) {
185                                 /* very close to 2pi */
186                                 x = p - PI2_L0;
187                                 return ((hx >= 0) ? x - PI2_L1 : -(x - PI2_L1));
100                                 return ((hx >= 0)? x - PI2_L1 : -(x - PI2_L1));
188                         }

190 #endif /* ! codereview */
191                         z = x * x;

193 #endif /* ! codereview */
194                         if (((ix - 0x40192000) >> 10) == 0) {
195                                 /* |x-2pi|<2**-8 */
196                                 w = (z * x) * (PP1 + z * PP2) - PI2_L;
197                         } else {
198                                 w = (z * x) * ((P1 + z * P2) + (z * z) * (P3 +
199                                     z * P4)) - PI2_L;
102                                 w = (z * x) * ((P1 + z * P2) +
103                                     (z * z) * (P3 + z * P4)) - PI2_L;
200                         }

202                         return ((hx >= 0) ? p + w : -p - w);
105                         return ((hx >= 0)? p + w : -p - w);
203                 }

205 #endif /* ! codereview */
206                 j <<= 1;
207                 w = _TBL_sincos[j + 1];
107                 w = _TBL_sincos[j+1];
208                 z = _TBL_sincos[j];
209                 p = v + (v * s) * (PP1 + s * PP2);
210                 q = s * (QQ1 + s * QQ2);
211                 v = w * p + z * q;
212                 return ((hx >= 0) ? z + v : -z - v);
112                 return ((hx >= 0)? z + v : -z - v);
213         }

215         if (ix >= 0x7ff00000)           /* sin(Inf or NaN) is NaN */
216                 return (x / x);

218         /* argument reduction needed */
219         n = __rem_pio2(x, y);

221 #endif /* ! codereview */
222         switch (n & 3) {
223         case 0:
224                 return (__k_sin(y[0], y[1]));
225         case 1:
226                 return (__k_cos(y[0], y[1]));
227         case 2:
228                 return (-__k_sin(y[0], y[1]));
229         default:
230                 return (-__k_cos(y[0], y[1]));
231         }
232 }
```

```
**********************************************************
   10713 Thu Jun  6 20:38:02 2019
new/usr/src/lib/libm/common/C/sincos.c
11210 libm should be cstyle(10NBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

#endif /* ! codereview */
/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2005 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __sincos = sincos

/* INDENT OFF */
/*
 * sincos(x,s,c)
 * Accurate Table look-up algorithm by K.C. Ng, 2000.
 *
 * 1. Reduce x to x>0 by cos(-x)=cos(x), sin(-x)=-sin(x).
 * 2. For 0<= x < 8, let i = (64*x chopped)-10. Let d = x - a[i], where
 *    a[i] is a double that is close to (i+10.5)/64 (and hence |d|< 10.5/64)
 *    and such that sin(a[i]) and cos(a[i]) is close to a double (with error
 *    less than 2**-8 ulp). Then
 *
 *      cos(x) = cos(a[i]+d) = cos(a[i])cos(d) - sin(a[i])*sin(d)
 *             = TBL_cos_a[i]*(1+QQ1*d^2+QQ2*d^4) -
 *                     TBL_sin_a[i]*(d+PP1*d^3+PP2*d^5)
 *             = TBL_cos_a[i] + (TBL_cos_a[i]*d^2*(QQ1+QQ2*d^2) -
 *                     TBL_sin_a[i]*(d+PP1*d^3+PP2*d^5))
 *
 *      sin(x) = sin(a[i]+d) = sin(a[i])cos(d) + cos(a[i])*sin(d)
 *             = TBL_sin_a[i]*(1+QQ1*d^2+QQ2*d^4) +
 *                     TBL_cos_a[i]*(d+PP1*d^3+PP2*d^5)
 *             = TBL_sin_a[i] + (TBL_sin_a[i]*d^2*(QQ1+QQ2*d^2) +
 *                     TBL_cos_a[i]*(d+PP1*d^3+PP2*d^5))
 *
 *    Note: for x close to n*pi/2, special treatment is need for either
 *    sin or cos:
 *    i in [81, 100] (  pi/2 +-10.5/64 => tiny cos(x) = sin(pi/2-x)
 *    i in [181,200] (  pi   +-10.5/64 => tiny sin(x) = sin(pi-x)
```



```
 *    i in [282,301] (  3pi/2+-10.5/64 => tiny cos(x) = sin(x-3pi/2)
 *    i in [382,401] (  2pi  +-10.5/64 => tiny sin(x) = sin(x-2pi)
 *    i in [483,502] (  5pi/2+-10.5/64 => tiny cos(x) = sin(5pi/2-x)
 *
 * 3. For x >= 8.0, use kernel function __rem_pio2 to perform argument
 *    reduction and call __k_sincos_ to compute sin and cos.
 *
 * kernel function:
 *      __rem_pio2      ... argument reduction routine
 *      __k_sincos_     ... sine and cosine function on [-pi/4,pi/4]
 *
 * Method.
 *      Let S and C denote the sin and cos respectively on [-PI/4, +PI/4].
 *      1. Assume the argument x is reduced to y1+y2 = x-k*pi/2 in
 *         [-pi/2 , +pi/2], and let n = k mod 4.
 *      2. Let S=S(y1+y2), C=C(y1+y2). Depending on n, we have
 *
 *          n        sin(x)      cos(x)        tan(x)
 *     ----------------------------------------------------------
 *          0          S           C             S/C
 *          1          C          -S            -C/S
 *          2         -S          -C             S/C
 *          3         -C           S            -C/S
 *     ----------------------------------------------------------
 *
 * Special cases:
 *      Let trig be any of sin, cos, or tan.
 *      trig(+-INF)  is NaN, with signals;
 *      trig(NaN)    is that NaN;
 *
 * Accuracy:
 *      TRIG(x) returns trig(x) nearly rounded (less than 1 ulp)
 */

#include "libm.h"

/* BEGIN CSTYLED */
#endif /* ! codereview */
static const double sc[] = {
/* ONE  = */
        1.0,
/* ONE  = */  1.0,
/* NONE = */ -1.0,

#endif /* ! codereview */
/*
 * |sin(x) - (x+pp1*x^3+pp2*x^5)| <= 2^-58.79 for |x| < 0.008
 */
/* PP1  = */-0.16666666666663165588672520523788895214806278586830555 67,
/* PP2  = */.00833331565299747232356489424846675824847537497797401 7927,

/* PP1  = */ -0.16666666666631655886725205237888952148062785868305556 7,
/* PP2  = */   .00833331565299747232356489424846675824847537497797401 7927,
/*
 * |(sin(x) - (x+p1*x^3+...+p4*x^9)|
 * |------------------------------ | <= 2^-57.63 for |x| < 0.1953125
 * |                 x             |
 */
/* P1   = */ -1.66666666666662969805215138920301589656e-0001,
/* P2   = */ 8.33333333239095129568399345528033637 6663e-0003,
/* P3   = */ -1.98412623799797669279155177823009840396 0e-0004,
/* P4   = */ 2.75340362485427723764998762284833035111 0e-0006,

#endif /* ! codereview */
/*
 * |cos(x) - (1+qq1*x^2+qq2*x^4)| <= 2^-55.99 for |x| <= 0.008 (0x3f80624d)
```

```
124  */
125 /* QQ1  = */-0.49999999999754923818429119819484185427429 29,
126 /* QQ2  = */0.04166654290435205929454520915835764039877 1740,
 97 /* QQ1  = */ -0.49999999999754923818429119819484185427429 29,
 98 /* QQ2  = */  0.04166654290435205929454520915835764039877 1740,
127 /* Q1   = */ -0.5,
128 /* Q2   = */ 4.16666666650035070368094552086074861744 5e-0002,
129 /* Q3   = */ -1.38888859643697221069426629057784869600 6e-0003,
130 /* Q4   = */ 2.47856307885858947367951951789295349219 2e-0005,
131 /* PIO2_H    = */ 1.57079632679489655799 9,
132 /* PIO2_L    = */ 6.12323399573676588613 0e-17,
133 /* PIO2_L0   = */ 6.12323399572792216556 4e-17,
134 /* PIO2_L1   = */ 8.84372056613570112025 5e-29,
135 /* PI_H      = */ 3.14159265358979311599796346 85,
136 /* PI_L      = */ 1.22464679914735317722606593227425e-16,
137 /* PI_L0     = */ 1.22464679914558443311283879205095e-16,
138 /* PI_L1     = */ 1.76874411322714022330000523373551737 6e-28,
139 /* PI3O2_H   = */ 4.71238898038468967399 7,
140 /* PI3O2_L   = */ 1.83697019872102976583 9e-16,
141 /* PI3O2_L0  = */ 1.83697019872039613358 7e-16,
142 /* PI3O2_L1  = */ 6.33632252474920114222 6e-29,
143 /* PI2_H     = */ 6.28318530717958623199592693 70,
144 /* PI2_L     = */ 2.44929359829470635445213186454850e-16,
145 /* PI2_L0    = */ 2.44929359829116886622567758410190e-16,
146 /* PI2_L1    = */ 3.53748822645428044660001046747103475 2e-28,
147 /* PI5O2_H   = */ 7.85398163397448278999 5,
148 /* PI5O2_L   = */ 3.06161699786838294306 5e-16,
149 /* PI5O2_L0  = */ 3.06161699786194159886 5e-16,
150 /* PI5O2_L1  = */ 6.44134420043364078198 2e-28,
151 };
152 /* END CSTYLED */
124 /* INDENT ON */

154 #define ONE                     sc[0]
155 #define PP1                     sc[2]
156 #define PP2                     sc[3]
157 #define P1                      sc[4]
158 #define P2                      sc[5]
159 #define P3                      sc[6]
160 #define P4                      sc[7]
161 #define QQ1                     sc[8]
162 #define QQ2                     sc[9]
163 #define Q1                      sc[10]
164 #define Q2                      sc[11]
165 #define Q3                      sc[12]
166 #define Q4                      sc[13]
167 #define PIO2_H                  sc[14]
168 #define PIO2_L                  sc[15]
169 #define PIO2_L0                 sc[16]
170 #define PIO2_L1                 sc[17]
171 #define PI_H                    sc[18]
172 #define PI_L                    sc[19]
173 #define PI_L0                   sc[20]
174 #define PI_L1                   sc[21]
175 #define PI3O2_H                 sc[22]
176 #define PI3O2_L                 sc[23]
177 #define PI3O2_L0                sc[24]
178 #define PI3O2_L1                sc[25]
179 #define PI2_H                   sc[26]
180 #define PI2_L                   sc[27]
181 #define PI2_L0                  sc[28]
182 #define PI2_L1                  sc[29]
183 #define PI5O2_H                 sc[30]
184 #define PI5O2_L                 sc[31]
185 #define PI5O2_L0                sc[32]
186 #define PI5O2_L1                sc[33]
```



```
187 #define PoS(x, z)               ((x * z) * (PP1 + z * PP2))
188 #define PoL(x, z)               ((x * z) * ((P1 + z * P2) + (z * z) * \
189         (P3 + z * P4)))
160 #define PoL(x, z)       ((x * z) * ((P1 + z * P2) + (z * z) * (P3 + z * P4)))

191 extern const double _TBL_sincos[], _TBL_sincosx[];

193 void
194 sincos(double x, double *s, double *c)
195 {
165 sincos(double x, double *s, double *c) {
196         double z, y[2], w, t, v, p, q;
197         int i, j, n, hx, ix, lx;

199         hx = ((int *)&x)[HIWORD];
200         lx = ((int *)&x)[LOWORD];
201         ix = hx & ~0x80000000;

203         if (ix <= 0x3fc50000) {         /* |x| < 10.5/64 = 0.164062500 */
204                 if (ix < 0x3e400000) {  /* |x| < 2**-27 */
205                         if ((int)x == 0)
206                                 *c = ONE;

208 #endif /* ! codereview */
209                         *s = x;
210                 } else {
211                         z = x * x;

213 #endif /* ! codereview */
214                         if (ix < 0x3f800000) {  /* |x| < 0.008 */
215                                 q = z * (QQ1 + z * QQ2);
216                                 p = PoS(x, z);
217                         } else {
218                                 q = z * ((Q1 + z * Q2) + (z * z) * (Q3 + z *
219                                     Q4));
177                                 q = z * ((Q1 + z * Q2) + (z * z) *
178                                     (Q3 + z * Q4));
220                                 p = PoL(x, z);
221                         }

223 #endif /* ! codereview */
224                         *c = ONE + q;
225                         *s = x + p;
226                 }

228 #endif /* ! codereview */
229                 return;
230         }

232         n = ix >> 20;
233         i = (((ix >> 12) & 0xff) | 0x100) >> (0x401 - n);
234         j = i - 10;

236 #endif /* ! codereview */
237         if (n < 0x402) {                /* |x| < 8 */
238                 x = fabs(x);
239                 v = x - _TBL_sincosx[j];
240                 t = v * v;
241                 w = _TBL_sincos[(j << 1)];
242                 z = _TBL_sincos[(j << 1) + 1];
181                 w = _TBL_sincos[(j<<1)];
182                 z = _TBL_sincos[(j<<1)+1];
243                 p = v + PoS(v, t);
244                 q = t * (QQ1 + t * QQ2);

246                 if ((((j - 81) ^ (j - 101)) | ((j - 282) ^ (j - 302)) | ((j -
```

```
                    483) ^ (j - 503)) | ((j - 181) ^ (j - 201)) | ((j - 382) ^
                    (j - 402))) < 0) {
                if ((((j - 81) ^ (j - 101)) |
                    ((j - 282) ^ (j - 302)) |
                    ((j - 483) ^ (j - 503)) |
                    ((j - 181) ^ (j - 201)) |
                    ((j - 382) ^ (j - 402))) < 0) {
                    if (j <= 101) {
                            /* near pi/2, cos(x) = sin(pi/2-x) */
                            t = w * q + z * p;
                            *s = (hx >= 0) ? w + t : -w - t;
                            *s = (hx >= 0)? w + t : -w - t;
                            p = PIO2_H - x;
                            i = ix - 0x3ff921fb;
                            x = p + PIO2_L;

                            if ((i | ((lx - 0x54442D00) & 0xffffff00)) ==
                                0) {
                            if ((i | ((lx - 0x54442D00) &
                                0xffffff00)) == 0) {
                                    /* very close to pi/2 */
                                    x = p + PIO2_L0;
                                    *c = x + PIO2_L1;
                            } else {
                                    z = x * x;

#endif /* ! codereview */
                                    if (((ix - 0x3ff92000) >> 12) == 0) {
                                            /* |pi/2-x|<2**-8 */
                                            w = PIO2_L + PoS(x, z);
                                    } else {
                                            w = PIO2_L + PoL(x, z);
                                    }

#endif /* ! codereview */
                                    *c = p + w;
                            }
                    } else if (j <= 201) {
                            /* near pi, sin(x) = sin(pi-x) */
                            *c = z - (w * p - z * q);
                            p = PI_H - x;
                            i = ix - 0x400921fb;
                            x = p + PI_L;

                            if ((i | ((lx - 0x54442D00) & 0xffffff00)) ==
                                0) {
                            if ((i | ((lx - 0x54442D00) &
                                0xffffff00)) == 0) {
                                    /* very close to pi */
                                    x = p + PI_L0;
                                    *s = (hx >= 0) ? x + PI_L1 : -(x +
                                        PI_L1);
                                    *s = (hx >= 0)? x + PI_L1 :
                                        -(x + PI_L1);
                            } else {
                                    z = x * x;

#endif /* ! codereview */
                                    if (((ix - 0x40092000) >> 11) == 0) {
                                            /* |pi-x|<2**-8 */
                                            w = PI_L + PoS(x, z);
                                    } else {
                                            w = PI_L + PoL(x, z);
                                    }

                                    *s = (hx >= 0) ? p + w : -p - w;
```



```
                                    *s = (hx >= 0)? p + w : -p - w;
                            }
                    } else if (j <= 302) {
                            /* near 3/2pi, cos(x)=sin(x-3/2pi) */
                            t = w * q + z * p;
                            *s = (hx >= 0) ? w + t : -w - t;
                            *s = (hx >= 0)? w + t : -w - t;
                            p = x - PI3O2_H;
                            i = ix - 0x4012D97C;
                            x = p - PI3O2_L;

                            if ((i | ((lx - 0x7f332100) & 0xffffff00)) ==
                                0) {
                            if ((i | ((lx - 0x7f332100) &
                                0xffffff00)) == 0) {
                                    /* very close to 3/2pi */
                                    x = p - PI3O2_L0;
                                    *c = x - PI3O2_L1;
                            } else {
                                    z = x * x;

#endif /* ! codereview */
                                    if (((ix - 0x4012D800) >> 9) == 0) {
                                            /* |3/2pi-x|<2**-8 */
                                            w = PoS(x, z) - PI3O2_L;
                                    } else {
                                            w = PoL(x, z) - PI3O2_L;
                                    }

#endif /* ! codereview */
                                    *c = p + w;
                            }
                    } else if (j <= 402) {
                            /* near 2pi, sin(x)=sin(x-2pi) */
                            *c = z - (w * p - z * q);
                            p = x - PI2_H;
                            i = ix - 0x401921fb;
                            x = p - PI2_L;

                            if ((i | ((lx - 0x54442D00) & 0xffffff00)) ==
                                0) {
                            if ((i | ((lx - 0x54442D00) &
                                0xffffff00)) == 0) {
                                    /* very close to 2pi */
                                    x = p - PI2_L0;
                                    *s = (hx >= 0) ? x - PI2_L1 : -(x -
                                        PI2_L1);
                                    *s = (hx >= 0)? x - PI2_L1 :
                                        -(x - PI2_L1);
                            } else {
                                    z = x * x;

#endif /* ! codereview */
                                    if (((ix - 0x40192000) >> 10) == 0) {
                                            /* |x-2pi|<2**-8 */
                                            w = PoS(x, z) - PI2_L;
                                    } else {
                                            w = PoL(x, z) - PI2_L;
                                    }

                                    *s = (hx >= 0) ? p + w : -p - w;
                                    *s = (hx >= 0)? p + w : -p - w;
                            }
                    } else {
                            /* near 5pi/2, cos(x) = sin(5pi/2-x) */
                            t = w * q + z * p;
```

```
358                                 *s = (hx >= 0) ? w + t : -w - t;
241                                 *s = (hx >= 0)? w + t : -w - t;
359                                 p = PI5O2_H - x;
360                                 i = ix - 0x401F6A7A;
361                                 x = p + PI5O2_L;

363                                 if ((i | ((lx - 0x29553800) & 0xffffff00)) ==
364                                     0) {
245                                 if ((i | ((lx - 0x29553800) &
246                                     0xffffff00)) == 0) {
365                                         /* very close to pi/2 */
366                                         x = p + PI5O2_L0;
367                                         *c = x + PI5O2_L1;
368                                 } else {
369                                         z = x * x;

371 #endif /* ! codereview */
372                                         if (((ix - 0x401F6A7A) >> 7) == 0) {
373                                                 /* |5pi/2-x|<2**-8 */
374                                                 w = PI5O2_L + PoS(x, z);
375                                         } else {
376                                                 w = PI5O2_L + PoL(x, z);
377                                         }

379 #endif /* ! codereview */
380                                         *c = p + w;
381                                 }
382                         }
383                 } else {
384                         *c = z - (w * p - z * q);
385                         t = w * q + z * p;
386                         *s = (hx >= 0) ? w + t : -w - t;
252                         *s = (hx >= 0)? w + t : -w - t;
387                 }

389 #endif /* ! codereview */
390                 return;
391         }

393         if (ix >= 0x7ff00000) {
394                 *s = *c = x / x;
395                 return;
396         }

398         /* argument reduction needed */
399         n = __rem_pio2(x, y);

401 #endif /* ! codereview */
402         switch (n & 3) {
403         case 0:
404                 *s = __k_sincos(y[0], y[1], c);
405                 break;
406         case 1:
407                 *c = -__k_sincos(y[0], y[1], s);
408                 break;
409         case 2:
410                 *s = -__k_sincos(y[0], y[1], c);
411                 *c = -*c;
412                 break;
413         default:
414                 *c = __k_sincos(y[0], y[1], s);
415                 *s = -*s;
416         }
417 }
```

```
**********************************************************
    5440 Thu Jun  6 20:38:02 2019
new/usr/src/lib/libm/common/C/sincospi.c
11210 libm should be cstyle(10NBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

/* INDENT OFF */
/*
 * void sincospi(double x, double *s, double *c)
 * *s = sin(pi*x); *c = cos(pi*x);
 *
 * Algorithm, 10/17/2002, K.C. Ng
 * ------------------------------
 * Let y = |4x|, z = floor(y), and n = (int)(z mod 8.0) (displayed in binary).
 *      1. If y == z, then x is a multiple of pi/4. Return the following values:
 *             ----------------------------------------------------
 *               n  x mod 2    sin(x*pi)    cos(x*pi)   tan(x*pi)
 *             ----------------------------------------------------
 *              000  0.00       +0 ___       +1 ___       +0
 *              001  0.25       +\/0.5       +\/0.5       +1
 *              010  0.50       +1 ___       +0 ___       +inf
 *              011  0.75       +\/0.5       -\/0.5       -1
 *              100  1.00       -0 ___       -1 ___       +0
 *              101  1.25       -\/0.5       -\/0.5       +1
 *              110  1.50       -1 ___       -0 ___       +inf
 *              111  1.75       -\/0.5       +\/0.5       -1
 *             ----------------------------------------------------
 *      2. Otherwise,
 *             ----------------------------------------------------
 *               n     t        sin(x*pi)    cos(x*pi)   tan(x*pi)
 *             ----------------------------------------------------
 *              000  (y-z)/4     sinpi(t)     cospi(t)    tanpi(t)
 *              001  (z+1-y)/4   cospi(t)     sinpi(t)    1/tanpi(t)
 *              010  (y-z)/4     cospi(t)    -sinpi(t)   -1/tanpi(t)
 *              011  (z+1-y)/4   sinpi(t)    -cospi(t)   -tanpi(t)
 *              100  (y-z)/4    -sinpi(t)    -cospi(t)    tanpi(t)
```



```
 *              101  (z+1-y)/4  -cospi(t)    -sinpi(t)    1/tanpi(t)
 *              110  (y-z)/4    -cospi(t)     sinpi(t)   -1/tanpi(t)
 *              111  (z+1-y)/4  -sinpi(t)     cospi(t)   -tanpi(t)
 *             ----------------------------------------------------
 *
 * NOTE. This program compute sinpi/cospi(t<0.25) by __k_sin/cos(pi*t, 0.0).
 * This will return a result with error slightly more than one ulp (but less
 * than 2 ulp). If one wants accurate result,  one may break up pi*t in
 * high (tpi_h) and low (tpi_l) parts and call __k_sin/cos(tip_h, tip_lo)
 * instead.
 */

#include "libm.h"
#include "libm_protos.h"
#include "libm_macros.h"
#include <math.h>
#if defined(__SUNPRO_C)
#include <sunmath.h>
#endif

/* BEGIN CSTYLEd */
static const double pi = 3.14159265358979323846,        /* 400921FB,54442D18 */
static const double
        pi      = 3.14159265358979323846,       /* 400921FB,54442D18 */
        sqrth_h = 0.707106781186547572737310929369414225220680236816406 25,
        sqrth_l = -4.83364665672645651859358442991279322134116601310 04e-17;
/* END CSTYLED */
/* INDENT ON */

void
sincospi(double x, double *s, double *c)
{
        double y, z, t;
        int n, ix, k;
        int hx = ((int *)&x)[HIWORD];
        unsigned h, lx = ((unsigned *)&x)[LOWORD];

        ix = hx & ~0x80000000;
        n = (ix >> 20) - 0x3ff;

#endif /* ! codereview */
        if (n >= 51) {                  /* |x| >= 2**51 */
                if (n >= 1024) {
#if defined(FPADD_TRAPS_INCOMPLETE_ON_NAN)
                        *s = *c = ix >= 0x7ff80000 ? x : x - x;
                        /* assumes sparc-like QNaN */
#else
                        *s = *c = x - x;
#endif
                } else {
                        if (n >= 53) {
                                *s = 0.0;
                                *c = 1.0;
                        } else if (n == 52) {
                                if ((lx & 1) == 0) {
                                        *s = 0.0;
                                        *c = 1.0;
                                } else {
                                        *s = -0.0;
                                        *c = -1.0;
                                }
                        } else {        /* n == 51 */
                                if ((lx & 1) == 0) {
                                        *s = 0.0;
                                        *c = 1.0;
                                } else {
```

```
                                        *s = 1.0;
                                        *c = 0.0;
                                }

#endif /* ! codereview */
                                if ((lx & 2) != 0) {
                                        *s = -*s;
                                        *c = -*c;
                                }
                        }
                }
        } else if (n < -2) {            /* |x| < 0.25 */
        } else if (n < -2)      /* |x| < 0.25 */
                *s = __k_sincos(pi * fabs(x), 0.0, c);
        } else {
        else {
                /* y = |4x|, z = floor(y), and n = (int)(z mod 8.0) */
                if (ix < 0x41C00000) {  /* |x| < 2**29 */
                        y = 4.0 * fabs(x);
                        n = (int)y;     /* exact */
                        z = (double)n;
                        k = z == y;
                        t = (y - z) * 0.25;
                } else {                /* 2**29 <= |x| < 2**51 */
                        y = fabs(x);
                        k = 50 - n;
                        n = lx >> k;
                        h = n << k;
                        ((unsigned *)&z)[LOWORD] = h;
                        ((int *)&z)[HIWORD] = ix;
                        k = h == lx;
                        t = y - z;
                }

#endif /* ! codereview */
                if (k) {                /* x = N/4 */
                        if ((n & 1) != 0) {
                                *s = *c = sqrth_h + sqrth_l;
                        } else {
                                if ((n & 2) == 0) {
                                        *s = 0.0;
                                        *c = 1.0;
                                } else {
                                        *s = 1.0;
                                        *c = 0.0;
                                }
                        }

#endif /* ! codereview */
                        if ((n & 4) != 0)
                                *s = -*s;

#endif /* ! codereview */
                        if (((n + 1) & 4) != 0)
                                *c = -*c;
                } else {
                        if ((n & 1) != 0)
                                t = 0.25 - t;

#endif /* ! codereview */
                        if (((n + (n & 1)) & 2) == 0)
                                *s = __k_sincos(pi * t, 0.0, c);
                        else
                                *c = __k_sincos(pi * t, 0.0, s);

#endif /* ! codereview */
```



```
                        if ((n & 4) != 0)
                                *s = -*s;

#endif /* ! codereview */
                        if (((n + 2) & 4) != 0)
                                *c = -*c;
                }
        }

#endif /* ! codereview */
        if (hx < 0)
                *s = -*s;
}
```

```
**********************************************************
    2053 Thu Jun  6 20:38:03 2019
new/usr/src/lib/libm/common/C/sinh.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
   1 /*
   2  * CDDL HEADER START
   3  *
   4  * The contents of this file are subject to the terms of the
   5  * Common Development and Distribution License (the "License").
   6  * You may not use this file except in compliance with the License.
   7  *
   8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
   9  * or http://www.opensolaris.org/os/licensing.
  10  * See the License for the specific language governing permissions
  11  * and limitations under the License.
  12  *
  13  * When distributing Covered Code, include this CDDL HEADER in each
  14  * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  15  * If applicable, add the following below this CDDL HEADER, with the
  16  * fields enclosed by brackets "[]" replaced with your own identifying
  17  * information: Portions Copyright [yyyy] [name of copyright owner]
  18  *
  19  * CDDL HEADER END
  20  */

  22 #endif /* ! codereview */
  23 /*
  24  * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
  25  */

  27 #endif /* ! codereview */
  28 /*
  29  * Copyright 2005 Sun Microsystems, Inc.  All rights reserved.
  30  * Use is subject to license terms.
  31  */

  33 #pragma weak __sinh = sinh


  21 /* INDENT OFF */
  36 /*
  37  * sinh(x)
  38  * Code originated from 4.3bsd.
  39  * Modified by K.C. Ng for SUN 4.0 libm.
  40  * Method :
  41  *      1. reduce x to non-negative by sinh(-x) = - sinh(x).
  42  *      2.
  43  *
  44  *                                   expm1(x) + expm1(x)/(expm1(x)+1)
  45  *   0 <= x <= lnovft   : sinh(x) := --------------------------------
  46  *                                                                    2
  47  *  lnovft <= x <  INF  : sinh(x) := exp(x-1024*ln2)*2**1023
  48  *
  49  *
  50  * Special cases:
  51  *      sinh(x) is x if x is +INF, -INF, or NaN.
  52  *      only sinh(0)=0 is exact for finite argument.
  53  *
  54  */
  41 /* INDENT ON */

  56 #include "libm.h"

  58 static const double ln2hi = 6.93147180369123816490e-01,
  45 static const double
```



```
  46         ln2hi = 6.93147180369123816490e-01,
  59         ln2lo = 1.90821492927058770002e-10,
  60         lnovft = 7.09782712893383973096e+02;

  62 double
  63 sinh(double x)
  64 {
  51 sinh(double x) {
  65         double ox, r, t;

  67         ox = x;
  68         r = fabs(x);

  70 #endif /* ! codereview */
  71         if (!finite(x))
  72                 return (x * r);

  74 #endif /* ! codereview */
  75         if (r < lnovft) {
  76                 t = expm1(r);
  77                 r = copysign((t + t / (1.0 + t)) * 0.5, x);
  78         } else {
  79                 if (r < 1000.0)
  80                         x = copysign(exp((r - 1024 * ln2hi) - 1024 * ln2lo), x);

  82 #endif /* ! codereview */
  83                 r = scalbn(x, 1023);
  84         }

  86 #endif /* ! codereview */
  87         if (!finite(r))
  88                 r = _SVID_libm_err(ox, ox, 25);

  90 #endif /* ! codereview */
  91         return (r);
  92 }
```

```
**********************************************************
    1222 Thu Jun  6 20:38:03 2019
new/usr/src/lib/libm/common/C/sqrt.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
   1 /*
   2  * CDDL HEADER START
   3  *
   4  * The contents of this file are subject to the terms of the
   5  * Common Development and Distribution License (the "License").
   6  * You may not use this file except in compliance with the License.
   7  *
   8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
   9  * or http://www.opensolaris.org/os/licensing.
  10  * See the License for the specific language governing permissions
  11  * and limitations under the License.
  12  *
  13  * When distributing Covered Code, include this CDDL HEADER in each
  14  * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  15  * If applicable, add the following below this CDDL HEADER, with the
  16  * fields enclosed by brackets "[]" replaced with your own identifying
  17  * information: Portions Copyright [yyyy] [name of copyright owner]
  18  *
  19  * CDDL HEADER END
  20  */

  22 #endif /* ! codereview */
  23 /*
  24  * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
  25  */

  27 #endif /* ! codereview */
  28 /*
  29  * Copyright 2005 Sun Microsystems, Inc.  All rights reserved.
  30  * Use is subject to license terms.
  31  */

  33 #pragma weak __sqrt = sqrt

  35 #include "libm.h"


  37 extern double __inline_sqrt(double);

  39 double
  40 sqrt(double x)
  41 {
  25 sqrt(double x) {
  42         double z = __inline_sqrt(x);

  44         if (isnan(x))
  45                 return (z);

  47         return ((x < 0.0) ? _SVID_libm_err(x, x, 26) : z);
  30         return ((x < 0.0)? _SVID_libm_err(x, x, 26) : z);
  48 }
```

```
**********************************************************
    1824 Thu Jun  6 20:38:03 2019
new/usr/src/lib/libm/common/C/tan.c
11210 libm should be cstyle(10NBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __tan = tan


/* INDENT OFF */
/*
 * tan(x)
 * Table look-up algorithm by K.C. Ng, November, 1989.
 *
 * kernel function:
 *      __k_tan         ... tangent function on [-pi/4,pi/4]
 *      __rem_pio2      ... argument reduction routine
 */
/* INDENT ON */

#include "libm.h"
#include "libm_protos.h"
#include <math.h>

double
tan(double x)
{
tan(double x) {
        double y[2], z = 0.0;
        int n, ix;

        /* high word of x */
        ix = ((int *)&x)[HIWORD];
        ix = ((int *) &x)[HIWORD];

        /* |x| ~< pi/4 */
```



```
        ix &= 0x7fffffff;

        if (ix <= 0x3fe921fb) {
        if (ix <= 0x3fe921fb)
                return (__k_tan(x, z, 0));
        }

        /* tan(Inf or NaN) is NaN */
        else if (ix >= 0x7ff00000) {
#if defined(FPADD_TRAPS_INCOMPLETE_ON_NAN)
                return (ix >= 0x7ff80000 ? x : x - x);  /* NaN */
                /* assumes sparc-like QNaN */
#else
                return (x - x);                         /* NaN */
#endif
        }

        /* argument reduction needed */
        else {
                n = __rem_pio2(x, y);
                return (__k_tan(y[0], y[1], n & 1));
        }
}
______unchanged_portion_omitted_
```

```
**********************************************************
    2489 Thu Jun  6 20:38:04 2019
new/usr/src/lib/libm/common/C/tanh.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __tanh = tanh


/* INDENT OFF */
/*
 * TANH(X)
 * RETURN THE HYPERBOLIC TANGENT OF X
 * code based on 4.3bsd
 * Modified by K.C. Ng for sun 4.0, Jan 31, 1987
 *
 * Method :
 *      1. reduce x to non-negative by tanh(-x) = - tanh(x).
 *      2.
 *          0      <  x <=  1.e-10 :  tanh(x) := x
 *                                                -expm1(-2x)
 *          1.e-10 <  x <=  1      :  tanh(x) := --------------
 *                                               expm1(-2x) + 2
 *                                                     2
 *          1      <= x <=  22.0   :  tanh(x) := 1 -  ---------------
 *                                                    expm1(2x) + 2
 *          22.0   <  x <= INF     :  tanh(x) := 1.
 *
 *      Note: 22 was chosen so that fl(1.0+2/(expm1(2*22)+2)) == 1.
 *
 * Special cases:
 *      tanh(NaN) is NaN;
 *      only tanh(0)=0 is exact for finite argument.
 */

#include "libm.h"
```



```
#include "libm_protos.h"
#include <math.h>

static const double one = 1.0, two = 2.0, small = 1.0e-10, big = 1.0e10;

static const double
        one = 1.0,
        two = 2.0,
        small = 1.0e-10,
        big = 1.0e10;
/* INDENT ON */

double
tanh(double x)
{
        double t, y, z;
        int signx;
        volatile double dummy __unused;

        if (isnan(x))
                return (x * x);         /* + -> * for Cheetah */

#endif /* ! codereview */
        signx = signbit(x);
        t = fabs(x);
        z = one;

#endif /* ! codereview */
        if (t <= 22.0) {
                if (t > one) {
                if (t > one)
                        z = one - two / (expm1(t + t) + two);
                } else if (t > small) {
                else if (t > small) {
                        y = expm1(-t - t);
                        z = -y / (y + two);
                } else {
                        /* raise the INEXACT flag for non-zero t */
                        dummy = t + big;
#ifdef lint
                        dummy = dummy;
#endif
                        return (x);
                }
        } else if (!finite(t)) {
        } else if (!finite(t))
                return (copysign(1.0, x));
        } else {
        else
                return ((signx != 0) ? -z + small * small : z - small * small);
        }
#endif /* ! codereview */

        return ((signx != 0) ? -z : z);
}
```

```
**********************************************************
    2208 Thu Jun  6 20:38:04 2019
new/usr/src/lib/libm/common/C/xpg6.h
11210 libm should be cstyle(10NBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

#endif /* ! codereview */
/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#ifndef _XPG6_H
#define _XPG6_H

/*
 * The bits in lib/libc/inc/xpg6.h fpgroup may use as per PSARC/2003/486.
 */

/*
 * If set, math library entry points present in SUSv2 deal with exceptional
 * cases as per SUSv3 spec where math_errhandling is set to MATH_ERREXCEPT;
 * otherwise they behave as per SUSv2 spec.
 */
#define _C99SUSv3_math_errexcept                        0x00000400

#endif /* ! codereview */
/*
 * If set, pow(+/-1,+/-Inf) & pow(1,NaN) return 1; otherwise NaN is returned.
 * Analogous comment applies to powf and powl.
 */
#define _C99SUSv3_pow_treats_Inf_as_an_even_int         0x00000080

#endif /* ! codereview */
/*
 * If set, logb(subnormal) returns (double) ilogb(subnormal); otherwise
 * logb(subnormal) returns logb(DBL_MIN).  Analogous comment applies to
 * logbf and logbl.
 */
#define _C99SUSv3_logb_subnormal_is_like_ilogb          0x00000040
```



```
#endif /* ! codereview */
/*
 * If set, ilogb(0/+Inf/-Inf/NaN) raises FE_INVALID as per SUSv3; otherwise
 * no exception is raised.  Analogous comment applies to ilogbf and ilogbl.
 */
#define _C99SUSv3_ilogb_0InfNaN_raises_invalid          0x00000020

/*
 * __xpg6 = _C99SUSv3_mode_OFF disables C99/SUSv3 standards conformance mode.
 */
#define _C99SUSv3_mode_OFF                              0xFFFF0000

#if !defined(_ASM)
extern unsigned int __xpg6;
#endif

#endif /* _XPG6_H */
```

```
**********************************************************
    6714 Thu Jun  6 20:38:05 2019
new/usr/src/lib/libm/common/LD/_TBL_cosl.c
11210 libm should be cstyle(10NBLD) clean
**********************************************************
   1 /*
   2  * CDDL HEADER START
   3  *
   4  * The contents of this file are subject to the terms of the
   5  * Common Development and Distribution License (the "License").
   6  * You may not use this file except in compliance with the License.
   7  *
   8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
   9  * or http://www.opensolaris.org/os/licensing.
  10  * See the License for the specific language governing permissions
  11  * and limitations under the License.
  12  *
  13  * When distributing Covered Code, include this CDDL HEADER in each
  14  * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  15  * If applicable, add the following below this CDDL HEADER, with the
  16  * fields enclosed by brackets "[]" replaced with your own identifying
  17  * information: Portions Copyright [yyyy] [name of copyright owner]
  18  *
  19  * CDDL HEADER END
  20  */

  22 /*
  23  * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
  24  */

  26 #endif /* ! codereview */
  27 /*
  28  * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
  29  * Use is subject to license terms.
  30  */

  32 /*
  33  * For i = 0L, ..., 75 let x(i) be the extended precision number
  34  * whose exponent is given by 0x3ffc + ((i + 8) >> 5) and whose
  35  * five most significant fraction bits are given by (i + 8) & 0x1f.
  36  * (The remaining fraction bits are zero and the integer bit is 1.)
  37  * Then _TBL_cosl_hi[i] := cos(x(i)) rounded to extended precisionL,
  38  * and _TBL_cosl_lo[i] ~ cos(x(i)) - _TBL_cosl_hi[i].
  39  */

  41 #include "libm.h"

  43 const long double _TBL_cosl_hi[] = {
  44         9.87817783816471944077341 33e-01L, 9.872023778548304904145380 1e-01L,
  45         9.865719083994975887306512 5e-01L, 9.859263850706614357556970 0e-01L,
  46         9.852658177182138161845186 0e-01L, 9.845902164215998060179876 9e-01L,
  47         9.838995914896639721964645 4e-01L, 9.831939534604930725370658 4e-01L,
  48         9.824733131012552574926229 0e-01L, 9.817376814080357763344156 2e-01L,
  49         9.809870696056691904691875 2e-01L, 9.802214891475680962214765 7e-01L,
  50         9.794409517155483600086077 2e-01L, 9.786454692196508678599109 5e-01L,
  51         9.778350537979597933459230 4e-01L, 9.770097178164173847849348 4e-01L,
  52         9.761694738686352767142138 9e-01L, 9.753143347757023264932643 7e-01L,
  53         9.744443135859889803759327 5e-01L, 9.735594235749481714366042 3e-01L,
  54         9.726596782449127526573064 2e-01L, 9.717450913248894676151751 2e-01L,
  55         9.708156767703494629444607 7e-01L, 9.698714487630153450125301 8e-01L,
  56         9.689124217106447841708905 0e-01L, 9.669500292306778220226097 5e-01L,
  57         9.649286191047710095798628 5e-01L, 9.628483147093796998836415 2e-01L,
  58         9.607092430155619030640937 2e-01L, 9.585115345812286272988642 1e-01L,
  59         9.562553235431752969640355 2e-01L, 9.539407476088947339712929 8e-01L,
  60         9.515679480481722021627255 5e-01L, 9.491370696844630276451000 6e-01L,
  61         9.466482608860533218232344 3e-01L, 9.441016735570043456556889 3e-01L,
```



```
  62         9.414974631278810686179844 8e-01L, 9.388357885462654886521427 5e-01L,
  63         9.361168122670552902775745 2e-01L, 9.333407002425484356566282 0e-01L,
  64         9.305076219123142911601558 0e-01L, 9.276177501928519096509491 4e-01L,
  65         9.246712614670360985149287 5e-01L, 9.216683355733519181609073 0e-01L,
  66         9.186091557949182678528138 3e-01L, 9.154939088483012285860605 8e-01L,
  67         9.123227848721178464891021 2e-01L, 9.090959774154310516695691 5e-01L,
  68         9.058136834259364207600460 9e-01L, 9.024761032379415049168788 8e-01L,
  69         8.990834405601384561926812 9e-01L, 8.956359024631706989183661 8e-01L,
  70         8.921336993669944047109614 2e-01L, 8.885770450280355433302081 9e-01L,
  71         8.849661565261432916900188 9e-01L, 8.813012542513405991602241 9e-01L,
  72         8.775825618903727161302860 7e-01L, 8.699847180584173884433577 3e-01L,
  73         8.621744799348805043449385 5e-01L, 8.541537542773853851438975 4e-01L,
  74         8.459244992310679544687476 7e-01L, 8.374887238505236852922041 0e-01L,
  75         8.288484876093257348135187 6e-01L, 8.200058998972340082401696 9e-01L,
  76         8.109631195052179022031077 5e-01L, 8.017223540984184505884396 8e-01L,
  77         7.922858596771785431346624 1e-01L, 7.826559400262727969263543 1e-01L,
  78         7.728349461524715447845873 5e-01L, 7.628252757105762505308171 9e-01L,
  79         7.526293724180664760693183 8e-01L, 7.422497254585013069707460 9e-01L,
  80         7.316888688738208863251121 0e-01L, 7.209493809456964180594658 3e-01L,
  25          9.878177838164719440773413 3e-01L,
  26          9.872023778548304904145380 1e-01L,
  27          9.865719083994975887306512 5e-01L,
  28          9.859263850706614357556970 0e-01L,
  29          9.852658177182138161845186 0e-01L,
  30          9.845902164215998060179876 9e-01L,
  31          9.838995914896639721964645 4e-01L,
  32          9.831939534604930725370658 4e-01L,
  33          9.824733131012552574926229 0e-01L,
  34          9.817376814080357763344156 2e-01L,
  35          9.809870696056691904691875 2e-01L,
  36          9.802214891475680962214765 7e-01L,
  37          9.794409517155483600086077 2e-01L,
  38          9.786454692196508678599109 5e-01L,
  39          9.778350537979597933459230 4e-01L,
  40          9.770097178164173847849348 4e-01L,
  41          9.761694738686352767142138 9e-01L,
  42          9.753143347757023264932643 7e-01L,
  43          9.744443135859889803759327 5e-01L,
  44          9.735594235749481714366042 3e-01L,
  45          9.726596782449127526573064 2e-01L,
  46          9.717450913248894676151751 2e-01L,
  47          9.708156767703494629444607 7e-01L,
  48          9.698714487630153450125301 8e-01L,
  49          9.689124217106447841708905 0e-01L,
  50          9.669500292306778220226097 5e-01L,
  51          9.649286191047710095798628 5e-01L,
  52          9.628483147093796998836415 2e-01L,
  53          9.607092430155619030640937 2e-01L,
  54          9.585115345812286272988642 1e-01L,
  55          9.562553235431752969640355 2e-01L,
  56          9.539407476088947339712929 8e-01L,
  57          9.515679480481722021627255 5e-01L,
  58          9.491370696844630276451000 6e-01L,
  59          9.466482608860533218232344 3e-01L,
  60          9.441016735570043456556889 3e-01L,
  61          9.414974631278810686179844 8e-01L,
  62          9.388357885462654886521427 5e-01L,
  63          9.361168122670552902775745 2e-01L,
  64          9.333407002425484356566282 0e-01L,
  65          9.305076219123142911601558 0e-01L,
  66          9.276177501928519096509491 4e-01L,
  67          9.246712614670360985149287 5e-01L,
  68          9.216683355733519181609073 0e-01L,
  69          9.186091557949182678528138 3e-01L,
  70          9.154939088483012285860605 8e-01L,
  71          9.123227848721178464891021 2e-01L,
```

```
         9.0909597741543105166956915e-01L,
         9.0581368342593642076004609e-01L,
         9.0247610323794150491687888e-01L,
         8.9908344056013845619268129e-01L,
         8.9563590246317069891836618e-01L,
         8.9213369936699440471096142e-01L,
         8.8857704502803554333020819e-01L,
         8.8496615652614329169001889e-01L,
         8.8130125425134059916022419e-01L,
         8.7758256189037271613028607e-01L,
         8.6998471805841738884335773e-01L,
         8.6217447993488050434493855e-01L,
         8.5415375427738538514389754e-01L,
         8.4592449923106795446874767e-01L,
         8.3748872385052368529220410e-01L,
         8.2884848760932573481351876e-01L,
         8.2000589989723400824016969e-01L,
         8.1096311950521790220310775e-01L,
         8.0172235409841845058843968e-01L,
         7.9228585967717854313466241e-01L,
         7.8265594002627279692635431e-01L,
         7.7283494615247154478458735e-01L,
         7.6282527571057625053081719e-01L,
         7.5262937241806647606931838e-01L,
         7.4224972545850130697074609e-01L,
         7.3168886873820886325112 10e-01L,
         7.2094938094569641805946583e-01L,
        7.1003388356607967499180972e-01L,
};

const long double _TBL_cosl_lo[] = {
        2.3161701550475222913914987e-20L, -1.8449479910096732184579231e-20L,
        2.6686158961121436032543157e-20L, -8.6377467693509323999412576e-21L,
        1.9776110020628332806497627e-20L, -3.5925805070704800589322274e-21L,
         2.3161701550475222913914987e-20L,
        -1.8449479910096732184579231e-20L,
         2.6686158961121436032543157e-20L,
        -8.6377467693509323999412576e-21L,
         1.9776110020628332806497627e-20L,
        -3.5925805070704800589322274e-21L,
        -1.8155190558460064943241466e-20L,
        -9.1900782344860461108346151e-21L,
        -5.2952188498928572418662889e-21L, 1.8052490350294447403358175e-22L,
        1.4237809112451219388907461e-22L, 2.6375298402937478119012648e-20L,
        -1.0076765547845230197228052e-20L, 2.4356732099577389276048253e-20L,
        -1.3951467830437376437362152e-20L, 1.7110854885636746562043992e-20L,
        9.7751412348794551526570426e-21L, -1.5984515732024779414075399e-20L,
        -2.6221693743524256098098490e-20L, 2.1708281645344702813143892e-20L,
        1.3606643184793342931047312e-20L, 4.0913737251026449191179388e-21L,
        3.0297735892921952471510043e-21L, -2.0186136916357220892889611e-20L,
         -5.2952188498928572418662889e-21L,
          1.8052490350294447403358175e-22L,
          1.4237809112451219388907461e-22L,
          2.6375298402937478119012648e-20L,
         -1.0076765547845230197228052e-20L,
          2.4356732099577389276048253e-20L,
         -1.3951467830437376437362152e-20L,
          1.7110854885636746562043992e-20L,
          9.7751412348794551526570426e-21L,
         -1.5984515732024779414075399e-20L,
         -2.6221693743524256098098490e-20L,
          2.1708281645344702813143892e-20L,
          1.3606643184793342931047312e-20L,
          4.0913737251026449191179388e-21L,
          3.0297735892921952471510043e-21L,
         -2.0186136916357220892889611e-20L,
```



```
         -2.6295048282251297741856903e-20L,
         -1.4268128384616571293099177e-20L, 1.2118148575499258442724515e-21L,
         1.6059569963428104840244296e-20L, 2.5656322072743666174102425e-21L,
         3.1051993049709377435678279e-21L, 1.1564422287617245178214769e-20L,
         1.0031811944878086819339264e-20L, -1.7237335190163247756143591e-20L,
         2.0747363423904458194504323e-20L, 2.2865077385189808827392339e-20L,
         -2.5671240384658541701793951e-20L, 2.6526752505060021072717663e-20L,
         -1.9564443985440576261207264e-20L, 1.6662891366649668957364366e-20L,
         -1.4268128384616571293099177e-20L,
          1.2118148575499258442724515e-21L,
          1.6059569963428104840244296e-20L,
          2.5656322072743666174102425e-21L,
          3.1051993049709377435678279e-21L,
          1.1564422287617245178214769e-20L,
          1.0031811944878086819339264e-20L,
         -1.7237335190163247756143591e-20L,
          2.0747363423904458194504323e-20L,
          2.2865077385189808827392339e-20L,
         -2.5671240384658541701793951e-20L,
          2.6526752505060021072717663e-20L,
         -1.9564443985440576261207264e-20L,
          1.6662891366649668957364366e-20L,
         -1.3289734577249155895809888e-21L,
         -1.0679012486769670465318810e-20L,
         -2.2918344926389240849631303e-20L,
         -1.2815734598986502345856155e-20L, 1.4504064768242345767590746e-20L,
         -1.2815734598986502345856155e-20L,
          1.4504064768242345767590746e-20L,
         -1.4988853557132440148049946e-20L,
         -2.2142847270523120702212966e-20L, 2.9274200155749021994272015e-21L,
         -2.2142847270523120702212966e-20L,
          2.9274200155749021994272015e-21L,
         -1.9187410072234352245854903e-20L,
         -1.5529430996486684056198058e-20L, 8.3043961792850937525987774e-21L,
         2.3863634821654097616646090e-20L, -1.7796180005854437467836689e-20L,
         1.2938828814644961764053094e-20L, -1.2599167110905505919738134e-20L,
         7.2776486597245992496949283e-21L, -2.0062284600282808092832087e-20L,
         -1.5529430996486684056198058e-20L,
          8.3043961792850937525987774e-21L,
          2.3863634821654097616646090e-20L,
         -1.7796180005854437467836689e-20L,
          1.2938828814644961764053094e-20L,
         -1.2599167110905505919738134e-20L,
          7.2776486597245992496949283e-21L,
         -2.0062284600282808092832087e-20L,
         -1.4004485599673539406695080e-20L,
         -1.4442131618989703782137918e-20L, 2.2223959244287650022010583e-20L,
         -1.4442131618989703782137918e-20L,
          2.2223959244287650022010583e-20L,
         -4.4575975223558432505505015e-22L,
         -9.0245930394257121787744934e-21L, 2.3149253152495269264191463e-20L,
         -3.3469699832521350974745777e-21L, 1.5380944635427999356502468e-20L,
         -1.3572945384913555811651506e-20L, 1.9052929123346841342486920e-20L,
         6.8389097769442269862154625e-21L, 4.4331336879906155675581769e-21L,
         2.6264491975559389159451170e-20L, -2.3718434730140290189643472e-20L,
         -1.4777051948748214572130603e-20L, 2.0601161465229389031848878e-20L,
         -9.0245930394257121787744934e-21L,
          2.3149253152495269264191463e-20L,
         -3.3469699832521350974745777e-21L,
          1.5380944635427999356502468e-20L,
         -1.3572945384913555811651506e-20L,
          1.9052929123346841342486920e-20L,
          6.8389097769442269862154625e-21L,
          4.4331336879906155675581769e-21L,
          2.6264491975559389159451170e-20L,
         -2.3718434730140290189643472e-20L,
```

```
173         -1.477705194874821457213060 3e-20L,
174          2.060116146522938903184887 8e-20L,
126         -1.327334202764942777891340 2e-20L,
127         -1.565304786935923858497351 5e-20L,
128         -1.768807863560285665365512 5e-20L,
129 };
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   29212 Thu Jun  6 20:38:05 2019
new/usr/src/lib/libm/common/LD/_TBL_ipio2l.c
11210 libm should be cstyle(10NBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

/*
 * Table of constants for 2/pi, used in __rem_pio2l (trigl) function.
 * By K.C. Ng, April 25, 1989
 */

#include "libm.h"

const int _TBL_ipio2l_inf[] = {         /* by DHBailey MP package */
        0xA2F983, 0x6E4E44, 0x1529FC, 0x2757D1, 0xF534DD, 0xC0DB62,
        0x95993C, 0x439041, 0xFE5163, 0xABDEBB, 0xC561B7, 0x246E3A,
        0x424DD2, 0xE00649, 0x2EEA09, 0xD1921C, 0xFE1DEB, 0x1CB129,
        0xA73EE8, 0x8235F5, 0x2EBB44, 0x84E99C, 0x7026B4, 0x5F7E41,
        0x3991D6, 0x398353, 0x39F49C, 0x845F8B, 0xBDF928, 0x3B1FF8,
        0x97FFDE, 0x05980F, 0xEF2F11, 0x8B5A0A, 0x6D1F6D, 0x367ECF,
        0x27CB09, 0xB74F46, 0x3F669E, 0x5FEA2D, 0x7527BA, 0xC7EBE5,
        0xF17B3D, 0x0739F7, 0x8A5292, 0xEA6BFB, 0x5FB11F, 0x8D5D08,
        0x560330, 0x46FC7B, 0x6BABF0, 0xCFBC20, 0x9AF436, 0x1DA9E3,
        0x91615E, 0xE61B08, 0x659985, 0x5F14A0, 0x68408D, 0xFFD880,
        0x4D7327, 0x310606, 0x1556CA, 0x73A8C9, 0x60E27B, 0xC08C6B,
        0x47C419, 0xC367CD, 0xDCE809, 0x2A8359, 0xC4768B, 0x961CA6,
        0xDDAF44, 0xD15719, 0x053EA5, 0xFF0705, 0x3F7E33, 0xE832C2,
        0xDE4F98, 0x327DBB, 0xC33D26, 0xEF6B1E, 0x5EF89F, 0x3A1F35,
        0xCAF27F, 0x1D87F1, 0x21907C, 0x7C246A, 0xFA6ED5, 0x772D30,
        0x433B15, 0xC614B5, 0x9D19C3, 0xC2C4AD, 0x414D2C, 0x5D000C,
        0x467D86, 0x2D71E3, 0x9AC69B, 0x006233, 0x7CD2B4, 0x97A7B4,
        0xD55537, 0xF63ED7, 0x1810A3, 0xFC764D, 0x2A9D64, 0xABD770,
        0xF87C63, 0x57B07A, 0xE71517, 0x5649C0, 0xD9D63B, 0x3884A7,
        0xCB2324, 0x778AD6, 0x23545A, 0xB91F00, 0x1B0AF1, 0xDFCE19,
        0xFF319F, 0x6A1E66, 0x615799, 0x47FBAC, 0xD87F7E, 0xB76522,
        0x89E832, 0x60BFE6, 0xCDC4EF, 0x09366C, 0xD43F5D, 0xD7DE16,
```



```
        0xDE3B58, 0x929BDE, 0x2822D2, 0xE88628, 0x4D58E2, 0x32CAC6,
        0x16E308, 0xCB7DE0, 0x50C017, 0xA71DF3, 0x5BE018, 0x34132E,
        0x621283, 0x014883, 0x5B8EF5, 0x7FB0AD, 0xF2E91E, 0x434A48,
        0xD36710, 0xD8DDAA, 0x425FAE, 0xCE616A, 0xA4280A, 0xB499D3,
        0xF2A606, 0x7F775C, 0x83C2A3, 0x883C61, 0x78738A, 0x5A8CAF,
        0xBDD76F, 0x63A62D, 0xCBBFF4, 0xEF818D, 0x67C126, 0x45CA55,
        0x36D9CA, 0xD2A828, 0x8D61C2, 0x77C912, 0x142604, 0x9B4612,
        0xC459C4, 0x44C5C8, 0x91B24D, 0xF31700, 0xAD43D4, 0xE54929,
        0x10D5FD, 0xFCBE00, 0xCC941E, 0xEECE70, 0xF53E13, 0x80F1EC,
        0xC3E7B3, 0x28F8C7, 0x940593, 0x3E71C1, 0xB3092E, 0xF3450B,
        0x9C1288, 0x7B20AB, 0x9FB52E, 0xC29247, 0x2F327B, 0x6D550C,
        0x90A772, 0x1FE76B, 0x96CB31, 0x4A1679, 0xE27941, 0x89DFF4,
        0x9794E8, 0x84E6E2, 0x973199, 0x6BED88, 0x365F5F, 0x0EFDBB,
        0xB49A48, 0x6CA467, 0x427271, 0x325D8D, 0xB8159F, 0x09E5BC,
        0x25318D, 0x3974F7, 0x1C0530, 0x010C0D, 0x68084B, 0x58EE2C,
        0x90AA47, 0x02E774, 0x24D6BD, 0xA67DF7, 0x72486E, 0xEF169F,
        0xA6948E, 0xF691B4, 0x5153D1, 0xF20ACF, 0x339820, 0x7E4BF5,
        0x6863B2, 0x5F3EDD, 0x035D40, 0x7F8985, 0x295255, 0xC06437,
        0x10D86D, 0x324832, 0x754C5B, 0xD4714E, 0x6E5445, 0xC1090B,
        0x69F52A, 0xD56614, 0x9D0727, 0x50045D, 0xDB3BB4, 0xC576EA,
        0x17F987, 0x7D6B49, 0xBA271D, 0x296996, 0xACCCC6, 0x5414AD,
        0x6AE290, 0x89D988, 0x50722C, 0xBEA404, 0x940777, 0x7030F3,
        0x27FC00, 0xA871EA, 0x49C266, 0x3DE064, 0x83DD97, 0x973FA3,
        0xFD9443, 0x8C860D, 0xDE4131, 0x9D3992, 0x8C70DD, 0xE7B717,
        0x3BDF08, 0x2B3715, 0xA0805C, 0x93805A, 0x921110, 0xD8E80F,
        0xAF806C, 0x4BFFDB, 0x0F9038, 0x761859, 0x15A562, 0xBBCB61,
        0xB989C7, 0xBD4010, 0x04F2D2, 0x277549, 0xF6B6EB, 0xBB22DB,
        0xAA140A, 0x2F2689, 0x768364, 0x333B09, 0x1A940E, 0xAA3A51,
        0xC2A31D, 0xAEEDAF, 0x12265C, 0x4DC26D, 0x9C7A2D, 0x9756C0,
        0x833F03, 0xF6F009, 0x8C402B, 0x99316D, 0x07B439, 0x15200C,
        0x5BC3D8, 0xC492F5, 0x4BADC6, 0xA5CA4E, 0xCD37A7, 0x36A9E6,
        0x9492AB, 0x6842DD, 0xDE6319, 0xEF8C76, 0x528B68, 0x37DBFC,
        0xABA1AE, 0x3115DF, 0xA1AE00, 0xDAFB0C, 0x664D64, 0xB705ED,
        0x306529, 0xBF5657, 0x3AFF47, 0xB9F96A, 0xF3BE75, 0xDF9328,
        0x3080AB, 0xF68C66, 0x15CB04, 0x0622FA, 0x1DE4D9, 0xA4B33D,
        0x8F1B57, 0x09CD36, 0xE9424E, 0xA4BE13, 0xB52333, 0x1AAAF0,
        0xA8654F, 0xA5C1D2, 0x0F3F0B, 0xCD785B, 0x76F923, 0x048B7B,
        0x721789, 0x53A6C6, 0xE26E6F, 0x00EBEF, 0x584A9B, 0xB7DAC4,
        0xBA66AA, 0xCFCF76, 0x1D02D1, 0x2DF1B1, 0xC1998C, 0x77ADC3,
        0xDA4886, 0xA05DF7, 0xF480C6, 0x2FF0AC, 0x9AECDD, 0xBC5C3F,
        0x6DDED0, 0x1FC790, 0xB6DB2A, 0x3A25A3, 0x9AAF00, 0x9353AD,
        0x0457B6, 0xB42D29, 0x7E804B, 0xA707DA, 0x0EAA76, 0xA1597B,
        0x2A1216, 0x2DB7DC, 0xFDE5FA, 0xFEDB89, 0xFDBE89, 0x6C76E4,
        0xFCA906, 0x70803E, 0x156E85, 0xFF87FD, 0x073E28, 0x336761,
        0x86182A, 0xEABD4D, 0xAFE7B3, 0x6E6D8F, 0x396795, 0x5BBF31,
        0x48D784, 0x16DF30, 0x432DC7, 0x356125, 0xCE70C9, 0xB8CB30,
        0xFD6CBF, 0xA200A4, 0xE46C05, 0xA0DD5A, 0x476F21, 0xD21262,
        0x845CB9, 0x496170, 0xE0566B, 0x015299, 0x375550, 0xB7D51E,
        0xC4F133, 0x5F6E13, 0xE4305D, 0xA92E85, 0xC3B21D, 0x3632A1,
        0xA4B708, 0xD4B1EA, 0x21F716, 0xE4698F, 0x77FF27, 0x80030C,
        0x2D408D, 0xA0CD4F, 0x99A520, 0xD3A2B3, 0x0A5D2F, 0x42F9B4,
        0xCBDA11, 0xD0BE7D, 0xC1DB9B, 0xBD17AB, 0x81A2CA, 0x5C6A08,
        0x17552E, 0x550027, 0xF0147F, 0x8607E1, 0x640B14, 0x8D4196,
        0xDEBE87, 0x2AFDDA, 0xB6256B, 0x34897B, 0xFEF305, 0x9EBFB9,
        0x4F6A68, 0xA82A4A, 0x5AC44F, 0xBCF82D, 0x985AD7, 0x95C7F4,
        0x8D4D0D, 0xA63A20, 0x5F57A4, 0xB13F14, 0x953880, 0x0120CC,
        0x86DD71, 0xB6DEC9, 0xF560BF, 0x11654D, 0x6B0701, 0xACB08C,
        0xD0C0B2, 0x485551, 0x0EFB1E, 0xC37295, 0x3B06A3, 0x3540C0,
        0x7BDC06, 0xCC45E0, 0xFA294E, 0xC8CAD6, 0x41F3E8, 0xDE647C,
        0xD8649B, 0x31BED9, 0xC397A4, 0xD45877, 0xC5E369, 0x13DAF0,
        0x3C3ABA, 0x461846, 0x5F7555, 0xF5BDD2, 0xC6926E, 0x5D2EAC,
        0xED440E, 0x423E1C, 0x87C461, 0xE9FD29, 0xF3D6E7, 0xCA7C22,
        0x35916F, 0xC5E008, 0x8DD7FF, 0xE26A6E, 0xC6FDB0, 0xC10893,
        0x745D7C, 0xB2AD6B, 0x9D6ECD, 0x7B723E, 0x6A11C6, 0xA9CFF7,
        0xDF7329, 0xBAC9B5, 0x5100B7, 0x0DB2E2, 0x24BA74, 0x607DE5,
        0x8AD874, 0x2C150D, 0x0C1881, 0x94667E, 0x162901, 0x767A9F,
```

```
        0xBEFDFD, 0xEF4556, 0x367ED9, 0x13D9EC, 0xB9BA8B, 0xFC97C4,
        0x27A831, 0xC36EF1, 0x36C594, 0x56A8D8, 0xB5A8B4, 0x0ECCCF,
        0x2D8912, 0x34576F, 0x89562C, 0xE3CE99, 0xB920D6, 0xAA5E6B,
        0x9C2A3E, 0xCC5F11, 0x4A0BFD, 0xFBF4E1, 0x6D3B8E, 0x2C86E2,
        0x84D4E9, 0xA9B4FC, 0xD1EEEF, 0xC9352E, 0x61392F, 0x442138,
        0xC8D91B, 0x0AFC81, 0x6A4AFB, 0xD81C2F, 0x84B453, 0x8C994E,
        0xCC2254, 0xDC552A, 0xD6C6C0, 0x96190B, 0xB8701A, 0x649569,
        0x605A26, 0xEE523F, 0x0F117F, 0x11B5F4, 0xF5CBFC, 0x2DBC34,
        0xEEBC34, 0xCC5DE8, 0x605EDD, 0x9B8E67, 0xEF3392, 0xB817C9,
        0x9B5861, 0xBC57E1, 0xC68351, 0x103ED8, 0x4871DD, 0xDD1C2D,
        0xA118AF, 0x462C21, 0xD7F359, 0x987AD9, 0xC0549E, 0xFA864F,
        0xFC0656, 0xAE79E5, 0x362289, 0x22AD38, 0xDC9367, 0xAAE855,
        0x382682, 0x9BE7CA, 0xA40D51, 0xB13399, 0x0ED7A9, 0x480569,
        0xF0B265, 0xA7887F, 0x974C88, 0x36D1F9, 0xB39221, 0x4A827B,
        0x21CF98, 0xDC9F40, 0x5547DC, 0x3A74E1, 0x42EB67, 0xDF9DFE,
        0x5FD45E, 0xA4677B, 0x7AACBA, 0xA2F655, 0x23882B, 0x55BA41,
        0x086E59, 0x862A21, 0x834739, 0xE6E389, 0xD49EE5, 0x40FB49,
        0xE956FF, 0xCA0F1C, 0x8A59C5, 0x2BFA94, 0xC5C1D3, 0xCFC50F,
        0xAE5ADB, 0x86C547, 0x624385, 0x3B8621, 0x94792C, 0x876110,
        0x7B4C2A, 0x1A2C80, 0x12BF43, 0x902688, 0x893C78, 0xE4C4A8,
        0x7BDBE5, 0xC23AC4, 0xEAF426, 0x8A67F7, 0xBF920D, 0x2BA365,
        0xB1933D, 0x0B7CBD, 0xDC51A4, 0x63DD27, 0xDDE169, 0x19949A,
        0x9529A8, 0x28CE68, 0xB4ED09, 0x209F44, 0xCA984E, 0x638270,
        0x237C7E, 0x32B90F, 0x8EF5A7, 0xE75614, 0x08F121, 0x2A9DB5,
        0x4D7E6F, 0x5119A5, 0xABF9B5, 0xD6DF82, 0x61DD96, 0x023616,
        0x9F3AC4, 0xA1A283, 0x6DED72, 0x7A8D39, 0xA9B882, 0x5C326B,
        0x5B2746, 0xED3400, 0x7700D2, 0x55F4FC, 0x4D5901, 0x8071E0,
        0xE13F89, 0xB295F3, 0x64A8F1, 0xAEA74B, 0x38FC4C, 0xEAB2BB,
        0x47270B, 0xABC3A7, 0x34BA60, 0x52DD34, 0xF8563A, 0xEB7E8A,
        0x31BB36, 0x5895B7, 0x47F7A9, 0x94C3AA, 0xD39225, 0x1E7F3E,
        0xD8974E, 0xBBA94F, 0xD8AE01, 0xE661B4, 0x393D8E, 0xA523AA,
        0x33068E, 0x1633B5, 0x3BB188, 0x1D3A9D, 0x4013D0, 0xCC1BE5,
        0xF862E7, 0x3BF28F, 0x39B5BF, 0x0BC235, 0x22747E, 0xA247C0,
        0xD52D1F, 0x19ADD3, 0x9094DF, 0x9311D0, 0xB42B25, 0x496DB2,
        0xE264B2, 0x5EF135, 0x3BC6A4, 0x1A4AD0, 0xAAC92E, 0x64E886,
        0x573091, 0x982CFB, 0x311B1A, 0x08728B, 0xBDCEE1, 0x60E142,
        0xEB641D, 0xD0BBA3, 0xE559D4, 0x597B8C, 0x2A4483, 0xF332BA,
        0xF84867, 0x2C8D1B, 0x2FA9B0, 0x50F3DD, 0xF9F573, 0xDB61B4,
        0xFE233E, 0x6C41A6, 0xEEA318, 0x775A26, 0xBC5E5C, 0xCEA708,
        0x94DC57, 0xE20196, 0xF1E839, 0xBE4851, 0x5D2D2F, 0x4E9555,
        0xD96EC2, 0xE7D755, 0x6304E0, 0xC02E0E, 0xFC40A0, 0xBBF9B3,
        0x7125A7, 0x222DFB, 0xF619D8, 0x838C1C, 0x6619E6, 0xB20D55,
        0xBB5137, 0x79E809, 0xAF9149, 0x0D73DE, 0x0B0DA5, 0xCE7F58,
        0xAC1934, 0x724667, 0x7A1A13, 0x9E26BC, 0x4555E7, 0x585CB5,
        0x711D14, 0x486991, 0x480D60, 0x56ADAB, 0xD62F64, 0x96EE0C,
        0x212FF3, 0x5D6D88, 0xA67684, 0x95651E, 0xAB9E0A, 0x4DDEFE,
        0x571010, 0x836A39, 0xF8EA31, 0x9E381D, 0xEAC8B1, 0xCAC96B,
        0x37F21E, 0xD505E9, 0x984743, 0x9FC56C, 0x0331B7, 0x3B8BF8,
        0x86E56A, 0x8DC343, 0x6230E7, 0x93CFD5, 0x6A8F2D, 0x733005,
        0x1AF021, 0xA09FCB, 0x7415A1, 0xD56B23, 0x6FF725, 0x2F4BC7,
        0xB8A591, 0x7FAC59, 0x5C55DE, 0x212C38, 0xB13296, 0x5CFF50,
        0x366262, 0xFA7B16, 0xF4D9A6, 0x2ACFE7, 0xF07403, 0xD4D604,
        0x6FD916, 0x31B1BF, 0xCBB450, 0x5BD7C8, 0x0CE194, 0x6BD643,
        0x4FD91C, 0xDF4543, 0x5F3453, 0xE2B5AA, 0xC9AEC8, 0x131485,
        0xF9D2BF, 0xBADB9E, 0x76F5B9, 0xAF15CF, 0xCA3182, 0x14B56D,
        0xE9FE4D, 0x50FC35, 0xF5AED5, 0xA2D0C1, 0xC96057, 0x192EB6,
        0xE91D92, 0x07D144, 0xAEA3C6, 0x343566, 0x26D5B4, 0x3161E2,
        0x37F1A2, 0x209EFF, 0x958E23, 0x493798, 0x35F4A6, 0x4BDC02,
        0xC2BE13, 0xBE80A0, 0x0B72A3, 0x115C5F, 0x1E1BD1, 0x0DB4D3,
        0x869E85, 0x96976B, 0x2AC91F, 0x8A26C2, 0x3070F0, 0x041412,
        0xFC9FA5, 0xF72A38, 0x9C6878, 0xE2AA76, 0x50CFE1, 0x559274,
        0x934E38, 0x0A92F7, 0x5533F0, 0xA63DB4, 0x399971, 0xE2B755,
        0xA98A7C, 0x008F19, 0xAC54D2, 0x2EA0B4, 0xF5F3E0, 0x60C849,
        0xFFD269, 0xAE52CE, 0x7A5FDD, 0xE9CE06, 0xFB0AE8, 0xA50CCE,
        0xEA9D3E, 0x3766DD, 0xB834F5, 0x0DA090,
};
```



```
#if 0
const int _TBL_ipio2l_66[] = {
        0xA2F983, 0x6E4E44, 0x152A00, 0x062BC4, 0x0DA276, 0xBED4C1,
        0xFDF905, 0x5CD5BA, 0x767CEC, 0x1F80D6, 0xC26053, 0x3A0070,
        0x107C2A, 0xF68EE9, 0x687B7A, 0xB990AA, 0x38DE4B, 0x96CFF3,
        0x92735E, 0x8B34F6, 0x195BFC, 0x27F88E, 0xA93EC5, 0x3958A5,
        0x3E5D13, 0x1C55A8, 0x5B4A8B, 0xA42E04, 0x12D105, 0x35580D,
        0xF62347, 0x450900, 0xB98BCA, 0xF7E8A4, 0xA2E5D5, 0x69BC52,
        0xF0381D, 0x1A0A88, 0xFE8714, 0x7F6735, 0xBB7D4D, 0xC6F642,
        0xB27E80, 0x6191BF, 0xB6B750, 0x52776E, 0xD60FD0, 0x607DCC,
        0x68BFAF, 0xED69FC, 0x6EB305, 0xD2557D, 0x25BDFB, 0x3E4AA1,
        0x84472D, 0x8B0376, 0xF77740, 0xD290DF, 0x15EC8C, 0x45A5C3,
        0x6181EF, 0xC5E7E8, 0xD8909C, 0xF62144, 0x298428, 0x6E5D9D,
        0xF9A9B4, 0xCDBD2F, 0xC083E7, 0x0D3957, 0xECA3B2, 0x96223C,
        0xC1080D, 0x087D47, 0x7D7576, 0xA614B1, 0x42A4B6, 0xAA173C,
        0xE217E5, 0xFDCD34, 0x279D5F, 0x39AACA, 0x1CA8DF, 0x8B6633,
        0x5C49E4, 0xB56803, 0x1E7938, 0x741FDC, 0x4CB19B, 0xCECC3B,
        0x921EB7, 0x7C0FC3, 0x361F23, 0xF9EE22, 0xBA4235, 0xA5FCA3,
        0xBD4680, 0xFCDF65, 0xFC96AD, 0x31C90C, 0x919EEB, 0xFE0FB7,
        0x75B4B0, 0x693961, 0x75BCAA, 0xEB6F39, 0xA343C0, 0xD16FF2,
        0x33DAD0, 0xC1E095, 0x053182, 0x11E4A1, 0x40F943, 0x32D314,
        0xAF1B98, 0xE1B05A, 0xE5F3AD, 0x6E633F, 0x363D14, 0xA3777C,
        0xC8C6EE, 0x001E18, 0x0D180C, 0xAA1369, 0xEDFBA2, 0x998A9D,
        0x16E799, 0x693B75, 0x90EF50, 0x938DD4, 0xFB7ACD, 0x67CEEB,
        0x249DE3, 0x9B9B52, 0xD8CDAC, 0xC31A54, 0x855FBF, 0x848591,
        0x0954B0, 0x946B8C, 0xA4C7B4, 0x9A9E51, 0xF20425, 0xAA2637,
        0xFC6657, 0x7D8625, 0x620B74, 0x8B578D, 0xEC9A05, 0xDEF24F,
        0x7F19B0, 0xFC2544, 0x1DA0F1, 0x23790C, 0xC4294D, 0x6D3C32,
        0x66FE56, 0xD45562, 0x66264F, 0xA24162, 0x13E930, 0xB0E7C0,
        0xFA0E97, 0xBFC62C, 0x0E663F, 0x90F33B, 0x55E73C, 0xD791F7,
        0xD3F00D, 0xAB01C7, 0x40CF8F, 0xA593BA, 0xE627D5, 0x4A8308,
        0x32DC06, 0x80C876, 0x1C3DB5, 0xB5489F, 0x632CDF, 0xB02517,
        0xD17EFA, 0x92570F, 0xFAED44, 0x8F8536, 0x27069B, 0xC014DC,
        0x997D48, 0x961D61, 0x7A960B, 0x31B622, 0xD3C425, 0xA69520,
        0x98D29E, 0xF1C973, 0x5483D7, 0x99611E, 0xEAFF5F, 0x7DEFF1,
        0x98475C, 0x91C787, 0x537E17, 0x068C65, 0xF05E52, 0x942F04,
        0x37CF92, 0xEF4223, 0xC4C52F, 0x521DAA, 0xBAAF97, 0x972236,
        0xA2B3D3, 0x62C921, 0x8D3A8B, 0x2B3302, 0x6061B9, 0x0CBE94,
        0x75F451, 0xBD06DE, 0x86042D, 0xFB61ED, 0x4C8869, 0x590232,
        0x479963, 0x23518D, 0xAF5D28, 0x60C9DE, 0x473DB0, 0x9DE009,
        0xD8FC4C, 0xE96991, 0x9CA455, 0x800BC8, 0x977CE0, 0xDCBFA6,
        0x19D249, 0xA0F76D, 0x5F9B2F, 0x452BB3, 0x77E091, 0xB6383A,
        0x7BE9C2, 0x4BF7C1, 0x8A5EBF, 0xEB0D55, 0x9AF4DC, 0x275CA0,
        0xED09D0, 0xE50A7F, 0xBEF42C, 0x4803AF, 0x56139F, 0xD58848,
        0x797D96, 0xB8352E, 0x49D90D, 0x7607E0, 0xC99256, 0x75F530,
        0xB72237, 0x1AF080, 0xC2E813, 0x06CFA9, 0xB9DF8E, 0x919C38,
        0x89D97E, 0x0464D5, 0xB12EEF, 0xD14165, 0x365A72, 0x550D35,
        0x3772D8, 0xF41B58, 0x0378A7, 0x2D5D7D, 0xD6E433, 0xDD2018,
        0x139FD7, 0x1B5621, 0x94E046, 0x97A323, 0x693176, 0x28DF59,
        0xD24273, 0x0E4E26, 0xA9A8F6, 0xF15B41, 0x450EE3, 0x57EA61,
        0x7DADA6, 0xF21086, 0x394BEE, 0x8F4813, 0x3FDEE9, 0xF3A53D,
        0xAB2F40, 0x8B1E2B, 0xA07FD4, 0x992CC4, 0x63532D, 0x9F35A2,
        0x6FA290, 0x0094DE, 0xD2A24D, 0x755B81, 0x79F9E1, 0xFE1D35,
        0xFEE8CC, 0x9224C5, 0x54E2CE, 0x41F31C, 0xF45138, 0xED6D10,
        0x6B439D, 0xD2BE46, 0xC327D4, 0x68BFB0, 0x46D5A5, 0x79B285,
        0x776D7C, 0xE18647, 0x00E32F, 0xEBB7F2, 0x5DE307, 0x5A8EA0,
        0x06CEFE, 0x20923C, 0x354CE1, 0xAD09C5, 0x56996D, 0xCFB124,
        0xEF7BC1, 0x76BF72, 0xF20753, 0x5BBAFA, 0xB8A2B2, 0x5914F2,
        0x5D834F, 0xE64A08, 0x14C3AB, 0x07796B, 0xF2212D, 0xC74049,
        0xB61C6A, 0x282CFC, 0x25070C, 0x315BF1, 0x6FEAD3, 0x2CD2E5,
        0xD10F9C, 0x1972BB, 0x908073, 0x0F368C, 0x69BE97, 0xA242B0,
        0x722DFE, 0xAFE6A2, 0x143D8B, 0x5C5699, 0x48232B, 0xFF49AC,
        0xB5FA62, 0x6AD778, 0x7A844D, 0x258AA0, 0x8EDE3D, 0x9A9496,
        0x49924E, 0xA33E97, 0x4F43FA, 0xC40741, 0x2F764A, 0x8EB2B1,
        0x8E67D3, 0x9FF324, 0x51B11B, 0x5D6E09, 0xE9AD3E, 0xFFA902,
```

```
        0xF48653, 0x0845D3, 0xDED33E, 0x32D30E, 0x6247CA, 0x7C586D,
        0x2EAF9E, 0x323A35, 0xAD11FB, 0x0F420C, 0x0E0685, 0x401B60,
        0xBB3D43, 0xF4D489, 0xBCDC4C, 0x40FFBA, 0x18AB08, 0x7AC72D,
        0x5E76DB, 0xE8344E, 0x3975A2, 0xF9611B, 0x1121F3, 0x3A429C,
        0x9B18EC, 0xF298B1, 0x8AEC78, 0x1C248B, 0x69108F, 0xDB2D37,
        0xA1A613, 0x910359, 0x521451, 0xD4441F, 0x0BB3B6, 0x50D9DB,
        0xBD589F, 0x62A62E, 0xA9B903, 0x935F63, 0x058BEC, 0x78BCB5,
        0x2CB460, 0x3A9037, 0x0291C4, 0x1FABC1, 0xBE7D05, 0xF948E7,
        0x6BA5CD, 0xF62A0A, 0x9AEA19, 0x2257AB, 0x2E0D7D, 0x9EB93F,
        0x5E3F77, 0xD4A13F, 0x08E3DB, 0xDFD689, 0x2B9B4E, 0xB58427,
        0x25424B, 0x1197FD, 0xCF298A, 0x314008, 0xD5687F, 0x0F0EAC,
        0x13C485, 0xF684B2, 0xED7EC7, 0x6E636D, 0x28C933, 0xE19058,
        0x688B6A, 0xC88905, 0xFB2F31, 0x61304C, 0xC19765, 0x60D81A,
        0x57F276, 0xC6EFC4, 0x048954, 0x303470, 0xDA6F6F, 0x93901A,
        0x911439, 0x363D12, 0x59E72B, 0x6F9F1E, 0x57C584, 0xDF0D23,
        0xBB743F, 0xADE99C, 0x546097, 0xFCC820, 0xCBB968, 0xDA9B5F,
        0x0DC271, 0x563337, 0x9ED662, 0xE7C44F, 0x3129F8, 0xF5EAF9,
        0xDAF7F2, 0xCD09FF, 0xA92535, 0x441C29, 0x7DF436, 0xE2B00A,
        0x36746F, 0xF1DC61, 0x9D3C9C, 0x63AB71, 0xB8F3BB, 0x1C80F6,
        0x62FF65, 0x5FFE5F, 0x3B2814, 0xBADE27, 0x1B384B, 0x268AA9,
        0xBD91EF, 0xCA436B, 0xABE107, 0x88DCA6, 0xC3AFC0, 0x85D155,
        0x464A48, 0xBFDAEB, 0xC6F389, 0x907C11, 0x0D3E41, 0xCD2197,
        0x549008, 0x817E4E, 0x8C7154, 0x1DC37F, 0x5E897E, 0xA9A2FE,
        0xEC6060, 0xCC0728, 0x430D3B, 0x62471C, 0xD3A4D3, 0x2BA57B,
        0xE5D15A, 0xD632F3, 0xF2B76F, 0xEC8498, 0xAE41C2, 0xAAF413,
        0xEAF5C0, 0xDD1B07, 0xB9A2A0, 0x59F230, 0xA3F61B, 0x8F8643,
        0x05DE6B, 0x1B5B8E, 0x63ECC5, 0xBFF01D, 0x8F1440, 0x3F8ADF,
        0x2E6539, 0xF3DB7A, 0x293FE5, 0x7EE714, 0x88E6D8, 0x2B2A6A,
        0xDF6E34, 0x8D4604, 0x4F6594, 0x639063, 0x6B51CC, 0x0D05CD,
        0x009607, 0xE7BF70, 0xC9A0EA, 0x0D80DD, 0xA1A065, 0x0DCB8F,
        0xA48430, 0x715934, 0x6FC8E4, 0x6FFC52, 0xEF8B05, 0xDE506A,
        0xE62BBC, 0x31480F, 0xEA64EA, 0x51E6FB, 0x9AE773, 0x21C54D,
        0xBFA080, 0x273D1E, 0x9FFD4E, 0x0C2CA8, 0x0690A5, 0xF8773B,
        0x4B2680, 0x6E3F56, 0xC8B89F, 0x0B7BD0, 0x71C8BF, 0x5AABD3,
        0x2BA93E, 0x9D2EE1, 0xCDF2FA, 0xEE57BE, 0x84A116, 0xDA756D,
        0x8FD6C0, 0x927153, 0xFF5EF3, 0x9F8331, 0x713411, 0xF945F3,
        0x0382B2, 0x8BAE30, 0xBC45A4, 0x630101, 0x5C9C3A, 0x643CFD,
        0x48115C, 0x17F03E, 0xB5F55E, 0x288DAF, 0x725660, 0xFB58E0,
        0xFC189E, 0x1ECA69, 0xFB19A6, 0xFA7A92, 0x7CC48E, 0x869372,
        0x58089A, 0x16DB5C, 0xADC0CD, 0x09D3D4, 0xD1108E, 0xDC64ED,
        0x3A999C, 0xAA8716, 0x5A3D8E, 0x7037FB, 0x1976AD, 0xE477D7,
        0x23782B, 0xC51F39, 0x4A5E9A, 0xDAD9DA, 0xE5B559, 0x08EF06,
        0x76E24F, 0x7361AD, 0x5F42A3, 0x9B70E5, 0xCE96C4, 0x552E99,
        0x6D7A6F, 0x804474, 0x4FA45B, 0x1D115B, 0x6D109E, 0x0A1A63,
        0x1084A6, 0xE18E5D, 0x2D8589, 0x203345, 0x4851AF, 0xA71EDC,
        0x03B6B1, 0x267970, 0xDEC908, 0x795BED, 0x7099B9, 0x209321,
        0x7FC2E7, 0x0F3E5E, 0xC7A4F4, 0x088129, 0x59AE63, 0x4E3251,
        0x344268, 0x79285D, 0x2B9494, 0xF1E2A2, 0xF7DA20, 0xDF6756,
        0xCA3BA3, 0x422489, 0xA2239C, 0x38724D, 0x2AC767, 0x601E9D,
        0xB47C6C, 0xA22481, 0xBBB655, 0x1EC0C4, 0xD84A97, 0xD449EE,
        0x162C9D, 0x782F29, 0xCEB4FA, 0xE317BC, 0x2FFDBD, 0xB342D2,
        0xB2CB19, 0x323AB9, 0x1AFF93, 0x13A8DF, 0x86B5A5, 0x5741D6,
        0xC54342, 0x3CAC29, 0xF7517C, 0x129A7A, 0xB2B8B4, 0x9B709F,
        0x3923C5, 0xEAFA6E, 0xDB9077, 0x29EEA0, 0x702D8C, 0x4DC14F,
        0xE46933, 0xA764E4, 0x754266, 0xFA4F98, 0x643DA5, 0xCA775C,
        0x7F1632, 0xE671A3, 0x4BF4C6, 0xA82378, 0xEFD317, 0xE62D38,
        0xD461C9, 0x8EEC80, 0xC89882, 0x4CC73C, 0x830F3F, 0xE4B200,
        0x582615, 0x6CD558, 0xA66727, 0xEF7975, 0xFEA5CE, 0x147A40,
        0x4796E4, 0xC07761, 0xF5D5B3, 0x6B65FB, 0xE4F14D, 0xA837CA,
        0x9A152A, 0x554E94, 0x83EC5F, 0xA62174, 0x85E2ED, 0xCCE71C,
        0x3540FF, 0x088A84, 0xBA2816, 0x293610, 0x4C3EE7, 0x8E55A9,
        0x49E5E5, 0x782178, 0x45D2AA, 0x9BB449, 0x00D282, 0xF61E67,
        0xE2F7DE, 0xCC6AA1, 0xCD1979, 0x52FEDB, 0x9A8776, 0x70A018,
        0x500271, 0x1273BA, 0xDE648E, 0x7AC7F7, 0x767725, 0xD0A457,
        0xF17250, 0xBC578C, 0x2DFD3A, 0x97F988, 0xA576C8, 0x8129BB,
        0x22D9C3, 0x0436ED, 0x650791, 0xA314EC, 0x42A0B3, 0x37A521,
```



```
        0x4BFB2B, 0x8C1B7F, 0x115E17, 0xF7C27F, 0xC1D5EB, 0x060487,
        0x8A28D6, 0x41330F, 0xBFAE67, 0x7774E8, 0x4CCC3C, 0x6B2F80,
        0x628BF2, 0x1E41A6, 0x8D0B22, 0xBC85BA, 0xCCF461, 0xBEC69C,
        0xDF8A10, 0x3C5E71, 0x2F8D5F, 0x63D3DA, 0x5934D1, 0x2CA35D,
        0xC687A2, 0x24E9B4, 0x1843D3, 0x5C9B97, 0x9B580C, 0x780B2C,
        0x59943D, 0x0744D0, 0x8DA6E3, 0x07AAF6, 0x2214D0, 0x72E8D7,
        0x54151B, 0x514DE9, 0x8DCC3B, 0x0CEB00, 0x2C4DE3, 0x5012AE,
        0xD7B72E, 0xB7DE9A, 0x641B2F, 0xF9CF17, 0x8BD282, 0x9F31A3,
        0xDED846, 0x467E05, 0x26CCEA, 0xF8E404, 0x65572E, 0x82C594,
        0xE572A9, 0x895653, 0xA1AA94, 0x8DD876, 0x5E9A61, 0x69EB1C,
        0x0385A9, 0x5BC844, 0x95B2DF, 0x6678F6, 0xFA7033, 0xE4F434,
        0x5584A9, 0x32C099, 0x9AD846, 0xB3FFD1, 0xA81C56, 0x4E54EF,
        0x54D173, 0xF191B4, 0x49B2A2, 0xB309D9, 0x546D8D, 0xC0A51E,
        0xCAFFC0, 0x785400, 0x05F69D, 0x894056, 0xC33098, 0xDFF6C2,
        0x908D97, 0x05CC96, 0x46484B, 0xBD7B9D, 0xB152F5, 0x5A7461,
        0x59CA20, 0x8F8EF5, 0xC9FF05, 0xF6F398, 0x856C97, 0x81E07C,
        0xAE5EDA, 0x51BDC9, 0xF26437, 0xBBC8CE, 0x091B52, 0x68B6A5,
        0x90750E, 0x925EF9, 0x3D9CB3, 0x46EA96, 0x97D648, 0x78BCC7,
        0xF4B488, 0x05275E, 0x6619DF, 0x56D4A0, 0x8C5C41, 0xDB345A,
        0x0B79DA, 0x496369, 0x96109B, 0x667664, 0xC40CF9, 0x91D7CA,
        0x119F1A, 0xA99272, 0xCBB529, 0xBB033E, 0x8F91C0, 0x570045,
        0xB845C2, 0x2B8E52, 0x687AFB, 0x0D0AA3, 0x200863, 0x043B83,
        0xF129DE, 0x49C2D6, 0x9641D2, 0xC4747C, 0x220804, 0x503F05,
        0x7E274F, 0xCA83D9, 0x9D6495, 0x0E5039,
};

#endif /* ! codereview */
const int _TBL_ipio2l_53[] = {
        0xA2F983, 0x6E4E44, 0x16F3C4, 0xEA69B5, 0xD3E131, 0x60E1D2,
        0xD7982A, 0xC031F5, 0xD67BCC, 0xFD1375, 0x60919B, 0x3FA0BB,
        0x612ABB, 0x714F9B, 0x03DA8A, 0xC05948, 0xD023F4, 0x5AFA37,
        0x51631D, 0xCD7A90, 0xC0474A, 0xF6A6F3, 0x1A52E1, 0x5C3927,
        0x3ADA45, 0x4E2DB5, 0x64E8C4, 0x274A5B, 0xB74ADC, 0x1E6591,
        0x2822BE, 0x4771F5, 0x12A63F, 0x83BD35, 0x2488CA, 0x1FE1BE,
        0x42C21A, 0x682569, 0x2AFB91, 0x68ADE1, 0x4A42E5, 0x9BE357,
        0xB79675, 0xCE998A, 0x83AF8B, 0xE645E6, 0xDF0789, 0x9E9747,
        0xAA15FF, 0x358C3F, 0xAF3141, 0x72A3F7, 0x2BF1D4, 0xF3AD96,
        0x7D759F, 0x257FCE, 0x29FB69, 0xB1B42C, 0xC32DE1, 0x8C0BBD,
        0x31EC2F, 0x942026, 0x85DCE7, 0x653FF3, 0x136FA7, 0x0D7A5F,
        0x93FC61, 0x035287, 0xC77FCA, 0x73530A, 0xC6BC15, 0x0E4B0F,
        0x568FCE, 0x2D3456, 0x4D7FE1, 0xA12CD1, 0xB2CEA2, 0x531C62,
        0x70B4D2, 0x1BCE9A, 0x87704D, 0x6B83D7, 0xAA8121, 0x2530EA,
        0x2074BF, 0x28A071, 0x9D69C3, 0x406DD8, 0xF58783, 0x115D89,
        0x5E85F3, 0xAACDCC, 0x8C0B57, 0xD7DFFE, 0x550D96, 0xC43EB4,
        0x89ABA7, 0x94F595, 0x56F260, 0x06A4CD, 0x7FD2E2, 0x6FDFA8,
        0x3E9C98, 0xBFD682, 0xAD3A12, 0x23A8A6, 0x173A89, 0x5DE9BD,
        0x95A978, 0x28E484, 0x5964F3, 0x496AF0, 0x4B1DA9, 0x989061,
        0xBD2BF2, 0xE01A90, 0x0905B7, 0xAC39AC, 0x52D5B7, 0x109F25,
        0x3AE1DC, 0xF90A7C, 0x33F4E5, 0xF5DFDF, 0x1522D0, 0x562CE6,
        0x392CFF, 0xEB9032, 0x10A08E, 0x0B1D7F, 0x42B80A, 0x366DD2,
        0xC24F89, 0x02222E, 0x21494C, 0x985287, 0x87FD07, 0x2EE361,
        0xAD8D68, 0xE72273, 0x9E8D59, 0xD09999, 0x10F4A1, 0x1079A3,
        0xE9BEAF, 0x9C0887, 0x09C622, 0xEBCF06, 0x974532, 0x086A8F,
        0x6CEA05, 0x388C00, 0x74969E, 0xC85B16, 0x385A38, 0x9A2F35,
        0x670531, 0xABA6D0, 0xEFD3C1, 0x27AD92, 0xF4203E, 0x3D619F,
        0x4D05F4, 0x9AE7CC, 0x03B592, 0x41FF55, 0xCAFCA5, 0x1A0987,
        0x88AB79, 0x3627D4, 0x25B12A, 0x52594A, 0xA2BEB0, 0x25C3F2,
        0x4489DA, 0x7959A7, 0xEAEC89, 0xB34714, 0x960196, 0x1FC33A,
        0x7F0275, 0x32EF92, 0x0111CE, 0x8E4685, 0x6F5B34, 0xF6123A,
        0x5543B2, 0xE9A02A, 0x74E03F, 0x54D5A8, 0x086A2C, 0x4A9CD3,
        0x921191, 0x229764, 0x0A1A84, 0x9B45AE, 0xC653A5, 0xB15F33,
        0x100FD1, 0x7DD740, 0xB20CD3, 0x0A0786, 0xF506C3, 0x25EBF4,
        0x3AB39E, 0xE3BB24, 0x27646F, 0xEECE57, 0x706BFE, 0xC7A869,
        0x57ED51, 0x118C82, 0x2B0FF5, 0xC8E545, 0xC43D80, 0x2A3183,
        0x4C1BB9, 0xBC108A, 0x099779, 0xF9ECC8, 0x2A1063, 0x5D2F6A,
        0x8F2675, 0x12FF6D, 0x32EED9, 0xE4A245, 0x7392CF, 0x5C240B,
```

```
        0xC476FF, 0x97AFC7, 0xB76131, 0x665E05, 0x67BD57, 0x19E998,
        0x3A5863, 0x23B8AA, 0x5B5608, 0x8A66C6, 0x5F2AD3, 0x78BAFA,
        0x3516CE, 0xCBEA16, 0x6E40D4, 0xB463D4, 0xA6C12F, 0xABD3D7,
        0x32650A, 0x579D10, 0x3CB9E2, 0x1A02A7, 0xDF2FFA, 0x28C991,
        0xB2264C, 0x027870, 0x47BDD4, 0xF243B1, 0x39AE2C, 0x282EA4,
        0xAF1D98, 0x2AFD16, 0xABE7AF, 0x17CB67, 0x8FF93E, 0x793167,
        0x435F6B, 0x48058B, 0x417DA0, 0xE01217, 0x085A69, 0xB50E36,
        0x79A4CD, 0xD74907, 0x26C4B5, 0xB90054, 0x06C3AD, 0x5AB38F,
        0x585E91, 0xD04E4F, 0x2938CE, 0xD4EAA7, 0xA06DE5, 0x40BFE5,
        0xDE6849, 0xEF65F0, 0xF1D4BB, 0x94C21E, 0x66E978, 0x1B9B94,
        0x961043, 0x5961B8, 0xBAAA74, 0xD662EE, 0x9DABF6, 0x0AFE28,
        0x9587A4, 0xA632BC, 0x09149F, 0xDEA996, 0x2CAFD7, 0xBDE29B,
        0x7159E6, 0x1F7C49, 0xF2E2ED, 0xBFA992, 0x7C77EF, 0xC245D0,
        0xB2D129, 0x993E75, 0xAB4C0C, 0x5C84B6, 0x17F542, 0x45314E,
        0x1DEF1B, 0xE3BDCC, 0xB3AE86, 0x24522F, 0x918FC6, 0x2138D5,
        0x883646, 0x6858B6, 0x032762, 0x5170F8, 0x4974EA, 0x76BF77,
        0xECDA8A, 0x9EADDD, 0x2404EF, 0xC52A5D, 0xF2E858, 0xC42D60,
        0xD18C08, 0xDE59B2, 0x4CC3A6, 0x94D888, 0x4C4AF0, 0xCF1F8C,
        0xBF2F6F, 0x7B4535, 0x98B0DB, 0x2BE0CF, 0x4616A7, 0xA8D9FB,
        0x88CA7A, 0x5087E1, 0x18DD8A, 0x1A9F4F, 0x1DCECE, 0xF8609E,
        0xE2F0C8, 0x9AD7D4, 0xE3CDFE, 0xC6FDD5, 0x8FF3CD, 0x7D45AA,
        0xD34957, 0x7C1963, 0x6CE098, 0xB70215, 0x326BBF, 0x47B3A6,
        0xF9235D, 0x6F66F5, 0xC6E40C, 0xE7F50B, 0xFF2FDD, 0x5A1251,
        0xE95EF1, 0xDE8E67, 0xECEE9B, 0xC9F98E, 0x722224, 0x6DF750,
        0x81D08F, 0x2BFCF0, 0xDDC10D, 0x775314, 0xDB1D87, 0x41626B,
        0x9EDF31, 0x7738D9, 0x8D9EB4, 0x4F1C2A, 0xF3E795, 0xB69699,
        0xD9A56D, 0x31BB1B, 0x542975, 0xAB917B, 0x63927C, 0x9BB764,
        0x84A598, 0x0A0C51, 0x5E48C4, 0x7780E3, 0x87E156, 0x155972,
        0xE406F8, 0x48AB9E, 0x3CCDDA, 0x010F87, 0x683B70, 0x400CAD,
        0x5DE5C5, 0x7262FA, 0xFA248D, 0x013AF2, 0xE2E8B5, 0x995F7D,
        0x7F8C4B, 0x0E8B59, 0x1006F1, 0x40B6E9, 0x760654, 0xCBCC8C,
        0x086F40, 0xDC7F6F, 0xFCD0D4, 0xA47ADE, 0x5204FA, 0xF38A9D,
        0xE76C7C, 0x575207, 0x499BF1, 0x0DB01C, 0x09098E, 0x957A71,
        0xD53E0E, 0x61DF1D, 0xE6EF34, 0x5821EC, 0x96BCC0, 0xDC96CE,
        0xA9C0AE, 0x130B2C, 0xCCC589, 0x829BB9, 0x2A75BA, 0x97611C,
        0x0CEAB8, 0x165D9D, 0x35AD41, 0x82A805, 0x975628, 0x5601A6,
        0x074F08, 0x80A27D, 0xEFA64E, 0xD7BB4B, 0x5E6397, 0xC92FFC,
        0x4F3F7A, 0xBEA764, 0x0C9B7D, 0xC5DC74, 0xEAD216, 0x6DBBC0,
        0x913E3E, 0xABF50B, 0x95B24A, 0x3FC9C5, 0xE7BA15, 0x8C7F70,
        0xF81358, 0x774606, 0xCE8C0D, 0xB6B268, 0xB85BA6, 0xAC9B2E,
        0x1AAB05, 0x0C6C82, 0x6EC2AE, 0x606874, 0x8F60BF, 0x1FBC7B,
        0x58C97A, 0x448794, 0xBA48A0, 0x72E882, 0x6D3568, 0xE131FD,
        0x4745D0, 0x0BFA1E, 0x07B01D, 0x474D43, 0x59387E, 0x5B0AD5,
        0xC37A8C, 0x0474E8, 0x13D99D, 0x68A13C, 0xB69118, 0x89228C,
        0x6F7D83, 0x86D665, 0x5C7744, 0xDD183E, 0x1C2E17, 0x712F5E,
        0x4AACCB, 0xB69B68, 0xA1201F, 0x743C2B, 0xF6AD70, 0x92E024,
        0xF34FD8, 0x33712E, 0xFE1D73, 0x4471F0, 0x7D0526, 0x58AF47,
        0x7B11FE, 0x1FCE4F, 0x1356C9, 0x9CE3CA, 0xA843C0, 0x8EEA3C,
        0xABEEE4, 0xA5D495, 0xA407A4, 0x31BB4B, 0x0AA1E3, 0x518E7C,
        0xAA4A66, 0xD82CD8, 0x6EF8D2, 0x6F32E6, 0x1DC26B, 0x17AE59,
        0x4B683B, 0x8D48F7, 0xF4FBD8, 0xD4FE0A, 0xE961DE, 0x87BD37,
        0xE6CCD6, 0xCBD76D, 0x3E99DE, 0xB72E21, 0x54EB90, 0x6AB45D,
        0x600AFB, 0xA17B2F, 0xDA0421, 0xE6CA95, 0x35AAA2, 0x7D8FB1,
        0x3207BB, 0xBF82EE, 0x71F55F, 0xC661CB, 0xBD72A1, 0xBF5A64,
        0x6E39E8, 0x6C6DE2, 0x2BD178, 0xAF62A5, 0xA7D86E, 0xE7D0FE,
        0x84DB03, 0x67FDA2, 0x2D6809, 0x0F8B8F, 0x1B50E3, 0x234EF5,
        0x7325ED, 0x8F8F4C, 0xC1E426, 0x3066AD, 0x0759A4, 0xE03390,
        0x70CC9A, 0x524F77, 0xCDD489, 0x97DD24, 0xA81858, 0xF24513,
        0xA9C18E, 0x2A2F82, 0xC2C014, 0xB8E7F0, 0x934036, 0xD36E51,
        0xD9A089, 0xDBC587, 0xB30418, 0x969192, 0x0A5213, 0xE21841,
        0x2881EC, 0x9A293F, 0x0DF705, 0x85B497, 0xE430B9, 0xE90ECF,
        0xC15FDC, 0x9E8A7E, 0xC5472D, 0xB54FBD, 0x456AF2, 0xCA80B6,
        0xAE25FE, 0xA03B46, 0x6C6CFD, 0x78382A, 0x0E7877, 0x7F2D31,
        0x03C827, 0x61CF52, 0x339A2F, 0x2286A9, 0xE41DF0, 0x640F5C,
        0xBEF364, 0x010506, 0x6D2C21, 0x841EFF, 0x7F3B5D, 0xD98DC8,
        0x0F9421, 0xA25B0C, 0x4C2C44, 0x922392, 0xB98A8A, 0x6179B9,
```



```
        0xF7B419, 0x289AAF, 0xE92F47, 0x5E47A2, 0x82927F, 0xC7290E,
        0x6C925C, 0xBA5A3C, 0x8FB7F6, 0x9C4BEE, 0x02C529, 0x0CFCD7,
        0x5EBD8C, 0x7196E0, 0x4B917E, 0x6B9780, 0x6A1731, 0xA617FF,
        0x27A20D, 0x5A56A3, 0x43C4DB, 0xC62EA4, 0x637A84, 0x1C46F9,
        0x33C780, 0x61A278, 0x4915C9, 0xD6C776, 0x6A7C66, 0xD8DD0C,
        0xF87EB1, 0x124C43, 0x5B87E7, 0x097456, 0x3C2FA7, 0x307C4A,
        0x54267A, 0x30E34E, 0xC0CF98, 0xD75B19, 0xFADEDB, 0x12CBE8,
        0x29F24C, 0x579C7E, 0xBF3682, 0xDCB460, 0xAE08B3, 0xA524BC,
        0xC181C2, 0x5DAB90, 0x466602, 0x55345B, 0xA13941, 0x47D820,
        0x278066, 0x81B089, 0x165EFB, 0x4D27FD, 0x2BF9F4, 0x2E2FFB,
        0x6106B5, 0xE76806, 0x445A84, 0x0BDA0D, 0x49D7A4, 0x72650D,
        0xCDC55B, 0x3E16BC, 0x132F6F, 0x29E8FD, 0xE58428, 0x621E41,
        0x7D2AC4, 0xAB5697, 0xAC61EB, 0xE5DAF0, 0x654ED6, 0x8E77E3,
        0x0B2FBC, 0x2E63A3, 0xC8296A, 0x8B631F, 0x4ECCA6, 0x91859C,
        0x9E3E45, 0x0E3CC7, 0xC12454, 0xCCBCB6, 0x17979E, 0xD0D374,
        0xA489A2, 0xC6258F, 0xE8EF9E, 0x12EE26, 0xC614C2, 0x62E23E,
        0xCA8C5C, 0x409AC9, 0x511D05, 0xA88CE0, 0x195500, 0xF7144F,
        0x913BB7, 0x17D064, 0xF6C9CE, 0xAC5D11, 0xD0C313, 0xBCCCB6,
        0xAAD4FC, 0xE47B2C, 0xFE4362, 0xF2E712, 0x2D5EFF, 0x833822,
        0x58A1D7, 0x68377C, 0xE49B25, 0x22B179, 0x048796, 0x069400,
        0xE670D3, 0xD2CB85, 0x55FBE6, 0x67F281, 0xFE2DE0, 0x8CFAF2,
        0x9865BC, 0x210CD3, 0x86DD70, 0x43D00F, 0x55E279, 0x679252,
        0x8D4F58, 0xE17AC5, 0x6A6127, 0x1B0876, 0x5D8ED0, 0x701330,
        0xD5BD25, 0xC9A126, 0x57C571, 0xDC5C3F, 0xB6D34E, 0xB72383,
        0x001A9E, 0x7D36C0, 0x8151F6, 0x65D7C1, 0xE1F513, 0xCD372A,
        0xE69B0C, 0xD02685, 0x23C3EB, 0x3544CB, 0xF0BE31, 0x83F399,
        0xCB93F8, 0xFFC693, 0x908EC6, 0x8E5DE1, 0x315B7E, 0x67CE7B,
        0x40AAF7, 0x7FD285, 0x069B36, 0x03C00A, 0x13C7D5, 0x0DA14C,
        0x1EAAD4, 0x2B777F, 0x8E05C1, 0x5AD1AE, 0x60C398, 0xA4EA59,
        0x10BEED, 0x88F2FA, 0x69B941, 0xA54E70, 0xA817C3, 0xB96246,
        0xE8EEDC, 0x56D570, 0xBBEBB5, 0xD8F235, 0x201AB9, 0x9CC747,
        0x5BC2FB, 0xC877F3, 0x428CF6, 0x4EEF84, 0xBF85FD, 0xEE6D34,
        0x84C2DE, 0xC42F4C, 0x1A513B, 0x9AC41F, 0x87FFFA, 0x1CA431,
        0x714252, 0xC73FB9, 0x662D89, 0x3D83BA, 0xBDF046, 0x2E4F62,
        0x76B7C0, 0x81336C, 0xBE80A9, 0x4C9D72, 0x739A15, 0x47972C,
        0xA36A1B, 0xD31731, 0x54BA46, 0x2E8C72, 0xFEA5A5, 0x9A7E5F,
        0xC359ED, 0x8F0FFB, 0x1270DA, 0x5E9B08, 0xB0BFCB, 0x36974C,
        0x6CD8F9, 0xD02E1F, 0x1C3F2F, 0xFCF8F0, 0x4C2C6D, 0x0B2169,
        0x48B9CE, 0x42737D, 0xA8E974, 0x64062D, 0xA86C59, 0xEEC419,
        0x047C83, 0x996A23, 0xF2A4C8, 0x4BE1B8, 0x348286, 0xE84240,
        0x8337CB, 0xE55A2F, 0xC17750, 0xA4DA06, 0x64347F, 0x59A5A1,
        0xDFF53D, 0x62A571, 0xEECF3A, 0x886700, 0xC06DAF, 0x4E161F,
        0x12670E, 0xBDFE1A, 0xA72B38, 0x5BA22C, 0xFED227, 0x3FC814,
        0x150E5A, 0xE99B3A, 0x8EE9FC, 0xBC1845, 0x32373A, 0xBDA476,
        0xCEB88F, 0x7FAED3, 0xDB9116, 0x31CF72, 0x1A5136, 0xC4F362,
        0xDE4799, 0x768043, 0x386207, 0x8E5497, 0xB0EF6D, 0x6C57FB,
        0xF56664, 0xD24F05, 0xE0F702, 0x8A41EF, 0xA2EC53, 0x09731C,
        0x6157FE, 0xC5731C, 0xEF1A2E, 0x60EC10, 0xA67EFE, 0x486A73,
        0x8004F6, 0xC3F482, 0x63BA28, 0x107282,
};
#endif
```

```
**********************************************************
    6778 Thu Jun  6 20:38:06 2019
new/usr/src/lib/libm/common/LD/_TBL_sinl.c
11210 libm should be cstyle(10NBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

/*
 * For i = 0L, ..., 75 let x(i) be the extended precision number
 * whose exponent is given by 0x3ffc + ((i + 8) >> 5) and whose
 * five most significant fraction bits are given by (i + 8) & 0x1f.
 * (The remaining fraction bits are zero and the integer bit is 1.)
 * Then _TBL_sinl_hi[i] := sin(x(i)) rounded to extended precisionL,
 * and _TBL_sinl_lo[i] ~ sin(x(i)) - _TBL_sinl_hi[i].
 */

#include "libm.h"

const long double _TBL_sinl_hi[] = {
        1.5561499277355604121432509e-01L,
        1.5947245893184341994353297e-01L,
        1.6332749173661285085207024e-01L,
        1.6718003236480673437500555e-01L,
        1.7103002203139501927501524e-01L,
        1.7487740199027218989302670e-01L,
        1.7872211353515365937804412e-01L,
        1.8256409800047155539783929e-01L,
        1.8640329676226988454758749e-01L,
        1.9023965123909906176839606e-01L,
        1.9407310289290979115543571e-01L,
        1.9790359322994628465735775e-01L,
        2.0173106380163880472144652e-01L,
        2.0555545620549551765724079e-01L,
        2.0937671208599364370531084e-01L,
        2.1319477313546989061102989e-01L,
        2.1700958109501015675778940e-01L,
        2.2082107775533849055107655e-01L,
```



```
        2.2462920495770529235180901e-01L,
        2.2843390459477474541995223e-01L,
        2.3223511861151146241076006e-01L,
        2.3603278900606633373558587e-01L,
        2.3982685783066156443802536e-01L,
        2.4361726719247488600575847e-01L,
        2.4740395925452292959266856e-01L,
        2.5496596041587846749013231e-01L,
        2.6251239976915328146124702e-01L,
        2.7004281671858503154006088e-01L,
        2.7755675164633632592044860e-01L,
        2.8505374594054742458945975e-01L,
        2.9253334202332754361585744e-01L,
        2.9999508337868305117438275e-01L,
        3.0743851458038085066887951e-01L,
        3.1486318131974525087106269e-01L,
        3.2226863043338662567511427e-01L,
        3.2965440993086017192298214e-01L,
        3.3702006902225307624892253e-01L,
        3.4436515814569840820730424e-01L,
        3.5168922899481405922451731e-01L,
        3.5899183454606505366498749e-01L,
        3.6627252908604756136416898e-01L,
        3.7353086823869294642950362e-01L,
        3.8076640899239019207055991e-01L,
        3.8797870972702504604426484e-01L,
        3.9516733024093423623426119e-01L,
        4.0233183177777311122311904e-01L,
        4.0947177705329506611003562e-01L,
        4.1658673028204111924766885e-01L,
        4.2367625720393801036934428e-01L,
        4.3073992511080319721861361e-01L,
        4.3777730287275513286178799e-01L,
        4.4478796096452721142060563e-01L,
        4.5177147149168377657582618e-01L,
        4.5872740821673659236961014e-01L,
        4.6565534658516018269211988e-01L,
        4.7255486375130445115036980e-01L,
        4.7942553860420300028150759e-01L,
        4.9307868575392305727079882e-01L,
        5.0661145481425736764773474e-01L,
        5.2002054195372700474845132e-01L,
        5.3330267353602017331871271e-01L,
        5.4645460691920356440616155e-01L,
        5.5947313124736687740433047e-01L,
        5.7235506823450724037203458e-01L,
        5.8509727294046215482874185e-01L,
        5.9769663453870153121657086e-01L,
        6.1015007707579137127265265e-01L,
        6.2245456022234368301943030e-01L,
        6.3460708001526929683284300e-01L,
        6.4660466959115237050095826e-01L,
        6.5844439991056754159573505e-01L,
        6.7012338047316289465094724e-01L,
        6.8163876002333416675559724e-01L,
        6.9298772724631791026551897e-01L,
        7.0416751145453367277888060e-01L,
};

const long double _TBL_sinl_lo[] = {
        -4.4044420388485708604352042e-21L,
        -9.3658505779466794663857779e-22L,
        -5.2040678607071393508410817e-21L,
        -4.0395267481940078256007650e-21L,
        6.3327332576496468315469778e-21L,
        2.6586707822142093837984364e-21L,
```

```
        -2.687878745005074423734528 2e-21L,
        1.706363566230559525065423 7e-21L,
        4.792492128253855504545534 3e-21L,
        -4.410169106693930218301047 0e-21L,
        6.194860091544782283098049 6e-22L,
        -4.963841364974950225161897 1e-21L,
        3.591627159765154622792647 3e-21L,
        -4.077715032367371279775656 9e-22L,
        6.579913659977989860364766 0e-21L,
        5.043144180223627127959654 7e-21L,
        2.788696763680438370241209 4e-21L,
        1.779794191550709466456411 9e-21L,
        -1.380455439293963558382925 1e-21L,
        4.785598118761546662515263 1e-21L,
        3.170821139040699750300490 0e-21L,
        -1.515783404472565256987326 3e-22L,
        3.343894673168401920463190 3e-21L,
        6.472479805685587711117540 1e-21L,
        4.180142867195331469783970 0e-21L,
        -2.575736536701222748201602 3e-21L,
        -1.029739451577181029507403 2e-20L,
        1.269417963773565672246452 8e-20L,
        1.574851278101117956530867 3e-21L,
        -1.696718485920290579270552 1e-21L,
        8.844885865233133625173173 7e-21L,
        -1.113446896904034029324182 5e-20L,
        1.623447179102532142047180 4e-21L,
        -6.026373819605448465175129 1e-21L,
        1.263165229582264684341417 2e-20L,
        -8.664410168758276285369490 6e-21L,
        1.235922217492385939727135 8e-20L,
        -1.321982158724183150802798 1e-22L,
        -1.932411099899529692210129 1e-21L,
        1.272280883008921424091538 5e-20L,
        8.740370447978594029921246 6e-21L,
        -1.266386362934275101596621 9e-20L,
        -1.335920606520052563420848 7e-20L,
        6.814854782218765238272731 9e-21L,
        1.057145057340289219158225 7e-20L,
        -6.013441355202306312913002 4e-21L,
        1.265840545763240744721193 7e-20L,
        1.144359827513728479760891 2e-20L,
        -7.660292250364769324633001 1e-21L,
        -2.292087639462408062451267 8e-21L,
        -1.690164025767178828521433 6e-22L,
        1.245038344092697369828501 3e-20L,
        5.862568790931064336125278 2e-21L,
        7.684889120754001489153943 4e-21L,
        -1.092036372791246692453170 5e-20L,
        -3.818480276243524228043890 6e-21L,
        -8.219649874141686839943370 3e-21L,
        -5.662268740730506505601513 0e-21L,
        -5.438735743720910211787793 0e-21L,
        1.176238185774170938309759 7e-20L,
        1.041839175608057621886470 0e-20L,
        -2.811995833152472836989469 7e-21L,
        -1.948646477680843357596427 6e-20L,
        1.291913132045812277535232 2e-20L,
        -2.334253339527873749483645 7e-20L,
        2.207676314725380202022778 7e-20L,
        1.089746837254262163412662 2e-21L,
        2.249640020911799402065173 0e-20L,
        1.746690966262434693239438 3e-20L,
        2.308390244512709133606749 2e-20L,
        -6.151097811162159651983291 9e-21L,
        3.584342407584371543639495 3e-21L,
```



```
        -2.235528818100159779666199 4e-20L,
        1.629652187446452114094574 1e-20L,
        1.178911365589689956147755 9e-21L,
};
```

```
**********************************************************
    6783 Thu Jun  6 20:38:06 2019
new/usr/src/lib/libm/common/LD/_TBL_tanl.c
11210 libm should be cstyle(10NBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

/*
 * For i = 0L, ..., 75 let x(i) be the extended precision number
 * whose exponent is given by 0x3ffc + ((i + 8) >> 5) and whose
 * five most significant fraction bits are given by (i + 8) & 0x1f.
 * (The remaining fraction bits are zero and the integer bit is 1.)
 * Then _TBL_tanl_hi[i] := tan(x(i)) rounded to extended precisionL,
 * and _TBL_tanl_lo[i] ~ tan(x(i)) - _TBL_tanl_hi[i].
 */

#include "libm.h"

const long double _TBL_tanl_hi[] = {
        1.575341073252716106879028 9e-01L,
        1.615397840495214763138851 6e-01L,
        1.655505192739339762086122 5e-01L,
        1.695664452197665101484567 7e-01L,
        1.735876947679815208498048 7e-01L,
        1.776144014774467276340580 1e-01L,
        1.816466996033214276575276 6e-01L,
        1.856847241156344116200628 9e-01L,
        1.897286107180591328879096 2e-01L,
        1.937784958668918635222829 3e-01L,
        1.978345167902386688118772 7e-01L,
        2.018968115074171328874198 1e-01L,
        2.059655188485788721068853 5e-01L,
        2.100407784745589808458703 1e-01L,
        2.141227308969586648891396 4e-01L,
        2.182115174984674325041333 9e-01L,
        2.223072805534313308724976 2e-01L,
        2.264101632486738374742387 9e-01L,
```



```
        2.305203097045761414612919 9e-01L,
        2.346378649964236789960368 7e-01L,
        2.387629751760259202666330 0e-01L,
        2.428957872936165424056524 3e-01L,
        2.470364494200412646638396 0e-01L,
        2.511851106692407673926046 4e-01L,
        2.553419212210362665101993 9e-01L,
        2.636805964199967998440581 7e-01L,
        2.720536986587708834354516 8e-01L,
        2.804624701452514031732501 2e-01L,
        2.889081724405147259978048 8e-01L,
        2.973920872690245894751862 7e-01L,
        3.059155173530592641188783 5e-01L,
        3.144797872725715161626187 2e-01L,
        3.230862443517455201056308 4e-01L,
        3.317362595735727673438176 4e-01L,
        3.404312285238303874344671 7e-01L,
        3.491725723659103522444630 7e-01L,
        3.579617388480169938835076 1e-01L,
        3.668002033443234227315290 4e-01L,
        3.756894699317548404194060 8e-01L,
        3.846310725041492230356736 4e-01L,
        3.936265759256327582190238 7e-01L,
        4.026775772251402117857602 1e-01L,
        4.117857068341084757765509 9e-01L,
        4.209526298694758220878941 3e-01L,
        4.301800474642300490136365 1e-01L,
        4.394696981478662404705087 1e-01L,
        4.488233592792397088472831 9e-01L,
        4.582428485344323669688475 9e-01L,
        4.677300254523917999330360 3e-01L,
        4.772867930412522617102891 9e-01L,
        4.869150994484063245035503 8e-01L,
        4.966169396975656256997076 1e-01L,
        5.063943574962298120514109 2e-01L,
        5.162494471171751445125013 0e-01L,
        5.261843553577791441798125 5e-01L,
        5.362012835812160313660179 6e-01L,
        5.463024898437905132694315 8e-01L,
        5.667670655805864456805442 9e-01L,
        5.875973675914432214212324 0e-01L,
        6.088137403243807213907255 7e-01L,
        6.304376738358847668576567 8e-01L,
        6.524918979288079927056354 1e-01L,
        6.750004851442429076608525 7e-01L,
        6.979889636235992551524520 7e-01L,
        7.214844409909044199691839 6e-01L,
        7.455157405593919951237481 8e-01L,
        7.701135513442087050166133 5e-01L,
        7.953105935686741856231220 2e-01L,
        8.211418015898941218924309 0e-01L,
        8.476445264465526541089283 9e-01L,
        8.748587605544823495246423 2e-01L,
        9.028273874526735021757081 8e-01L,
        9.315964599440724611600570 0e-01L,
        9.612155104943704161620833 5e-01L,
        9.917378983632686802640772 4e-01L,
};

const long double _TBL_tanl_lo[] = {
        -2.677115940910573170140551 0e-21L,
        -4.609922678974126290021060 6e-21L,
        5.318664414037532282080245 8e-21L,
        2.513840583093863373568683 9e-21L,
        -5.131461705780643270699969 4e-21L,
        -2.315081845852432077193631 7e-21L,
```

```
        7.482315068840958985787834 6e-22L,
        6.5983384951777057330962451e-21L,
        3.1737465070309238679637904e-21L,
        -6.2605330413009742107992404e-21L,
        -3.4708968895421512574248288e-21L,
        -3.3508177722855547163047103e-21L,
        1.8539761255947162282442845e-21L,
        -4.3527863815358994574071238e-21L,
        -3.0729582373746958079080308e-21L,
        1.7486583794617176080777995e-21L,
        -2.0880427643688559927261666e-22L,
        3.4326156341633317484064051e-21L,
        -5.8444712515543005993510667e-21L,
        3.3308393583864583403400180e-21L,
        1.5180609545016167494014088e-21L,
        -4.5664864992230118395870971e-21L,
        -3.4486635382887607253671356e-22L,
        6.4992471510018586950169590e-21L,
        -5.7171552644357921603079772e-21L,
        1.0767820312749142840542796e-20L,
        -8.8873094864264944929118678e-21L,
        -3.6458345495736833933253427e-21L,
        3.7835691968285101289024150e-21L,
        -7.9922577212991920007926665e-21L,
        -1.1639426061963512311797196e-20L,
        1.0819496381458482697046145e-20L,
        1.2669812351932848585361942e-20L,
        -4.3879352642165387665557942e-21L,
        -6.2397232294970361376981025e-21L,
        1.0249894624181563425318369e-20L,
        4.8883545518509990780582976e-21L,
        -1.0924217224719888561366811e-20L,
        -1.0160304466598813882209781e-20L,
        5.1826415091471411711448075e-21L,
        1.0389918683332972349077236e-20L,
        -7.1664776574714262163862363e-21L,
        1.2298884220333748071625466e-20L,
        -1.3099990378137383497651040e-20L,
        6.6930911371536844477108605e-21L,
        1.3154437144468699485999317e-20L,
        -6.7276672708135125503950130e-21L,
        -9.6583948799780933132703713e-21L,
        -1.1693327591353762422287158e-20L,
        1.2115072030396340314945014e-20L,
        -4.8328734014430698289025015e-21L,
        1.0852973061445293626693228e-20L,
        1.9411831283588255256712679e-20L,
        -2.0725962316575506668083850e-20L,
        -9.1991091819589918968351350e-21L,
        -1.8439030785497371079388971e-20L,
        -1.4252114398617735096821730e-20L,
        -3.6634999903039053547935623e-22L,
        -2.7073538111310219812185487e-20L,
        2.1768400635771833866020006e-20L,
        -5.0453509036808273670769239e-21L,
        1.8262326404957249986102613e-20L,
        2.3253788272891224529527726e-21L,
        -2.6863465601726641017825874e-21L,
        2.0333919445169836552474035e-20L,
        1.2381983326738354735338055e-20L,
        -1.0629693225258909983165405e-20L,
        2.2479666845586239075466463e-20L,
        1.8993064919061156630226362e-20L,
        -1.8140078592138587341953739e-20L,
        1.5029592868184122759494625e-20L,
        2.0466189644006868146496769e-20L,
```



```
        5.1457594757697525471406575e-21L,
        2.3217272240793119168128789e-20L,
        -7.2198528398134119662230907e-21L,
};
```

```
**********************************************************
    4722 Thu Jun  6 20:38:07 2019
new/usr/src/lib/libm/common/LD/__cosl.c
11210 libm should be cstyle(10NBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

/* BEGIN CSTYLED */
/* INDENT OFF */
/*
 * __k_cosl( long double x;  long double y )
 * kernel cos function on [-pi/4, pi/4], pi/4 ~ 0.785398164
 * Input x is assumed to be bounded by ~pi/4 in magnitude.
 * Input y is the tail of x.
 *
 * Table look up algorithm
 *      1. by cos(-x) = cos(x), we may replace x by |x|
 *      2. if x < 25/128 = [0x3ffc4000, 0] = 0.15625 , then
 *           if x < 2^-57 (hx < 0x3fc60000 0), return 1.0 with inexact if x !=
 *           z = x*x;
 *           if x <= 1/128 = 2**-7 = 0.0078125
 *              cos(x)=1.0+z*(qq1+z*(qq2+z*(qq3+z*(qq4+z*qq5))))
 *           else
 *              cos(x)=1.0+z*(q1+ ... z*q8)
 *      3. else
 *              ht = (hx + 0x400)&0x7ffff800    (round x to a break point t)
 *              lt = 0
 *              i  = (hy-0x3ffc4000)>>11;       (i<=64)
 *              x' = (x - t)+y                  (|x'| ~<= 2^-7
 *         By
 *              cos(t+x')
 *                = cos(t)cos(x')-sin(t)sin(x')
 *                = cos(t)(1+z*(qq1+z*qq2))-[sin(t)]*x*(1+z*(pp1+z*pp2))
 *                = cos(t) + [cos(t)]*(z*(qq1+z*qq2))-
 *                              [sin(t)]*x*(1+z*(pp1+z*pp2))
 *
 *         Thus,
```



```
 *              let a= _TBL_cos_hi[i], b = _TBL_cos_lo[i], c= _TBL_sin_hi[i],
 *              x = (x-t)+y
 *              z = x*x;
 *              cos(t+x) = a+(b+ (-c*x*(1+z*(pp1+z*pp2))+a*(z*(qq1+z*qq2)))
 */
/* END CSTYLED */
#endif /* ! codereview */

#include "libm.h"

#include <sys/isa_defs.h>

extern const long double _TBL_cosl_hi[], _TBL_cosl_lo[], _TBL_sinl_hi[];
static const long double one = 1.0;

static const long double
one     = 1.0,
/*
 * |sin(x) - (x+pp1*x^3+...+ pp5*x^11 )| <= 2^-122.32 for |x|<1/64
 */
static const long double
        pp1 = -1.6666666666666666666666666586782940810e-0001L,
        pp2 = 8.3333333333333333333300372366092931754 0e-0003L,
        pp3 = -1.9841269841269840760459034837783378044 70e-0004L,
        pp4 = 2.7557319223619066413197231062109009494 13e-0006L,
        pp5 = -2.5051983985709470190939984691350120576 73e-0008L;

pp1     = -1.6666666666666666666666666586782940810e-0001L,
pp2     =  8.3333333333333333333300372366092931754 0e-0003L,
pp3     = -1.9841269841269840760459034837783378044 70e-0004L,
pp4     =  2.7557319223619066413197231062109009494 13e-0006L,
pp5     = -2.5051983985709470190939984691350120576 73e-0008L,
/*
 *
 * |cos(x) - (1+q1*x^2+...+q8*x^16)| <= 2^-117.11 for |x|<= 0.15625
 */
static const long double
        q1 = -4.9999999999999999999999999999999756416975e-0001L,
        q2 = 4.1666666666666666666666666406066577258e-0002L,
        q3 = -1.3888888888888888888888877700363937169637e-0003L,
        q4 = 2.4801587301587301584944684630318140835 59e-0005L,
        q5 = -2.7557319223985862763228192503560055428 71e-0007L,
        q6 = 2.0876756987674242614419597607298540178 55e-0009L,
        q7 = -1.1470744812396620890724521290107907747 61e-0011L,
        q8 = 4.7777616473996515997306634222635310347 82e-0014L;

q1      =  -4.9999999999999999999999999999999756416975e-0001L,
q2      =   4.1666666666666666666666666406066577258e-0002L,
q3      =  -1.3888888888888888888888877700363937169637e-0003L,
q4      =   2.4801587301587301584944684630318140835 59e-0005L,
q5      =  -2.7557319223985862763228192503560055428 71e-0007L,
q6      =   2.0876756987674242614419597607298540178 55e-0009L,
q7      =  -1.1470744812396620890724521290107907747 61e-0011L,
q8      =   4.7777616473996515997306634222635310347 82e-0014L,
/*
 *
 * |cos(x) - (1+qq1*x^2+...+ qq5*x^10)| <= 2^-123.84 for |x|<=1/128
 */
static const long double
        qq1 = -4.9999999999999999999999999999999378373641e-0001L,
        qq2 = 4.1666666666666666666666665478399327703130e-0002L,
        qq3 = -1.3888888888888888880582112306180516134 94e-0003L,
        qq4 = 2.4801587301561053777715856589053031118 66e-0005L,
        qq5 = -2.7557280997625263257364883766951570087 36e-0007L;
```

```
 85 qq1     =  -4.999999999999999999999999999999378373641e-0001L,
 86 qq2     =   4.166666666666666666666665478399327703130e-0002L,
 87 qq3     =  -1.388888888888888888058211230618051613494e-0003L,
 88 qq4     =   2.480158730156105377771585658905303111866e-0005L,
 89 qq5     =  -2.755728099762526325736488376695157008736e-0007L;
 90 /* INDENT ON */
112 long double
113 __k_cosl(long double x, long double y)
114 {
 92 __k_cosl(long double x, long double y) {
115         long double a, t, z, w;
116         int *pt = (int *)&t, *px = (int *)&x;
 94         int *pt = (int *) &t, *px = (int *) &x;
117         int i, j, hx, ix;

119         t = 1.0;
120 #if defined(__i386) || defined(__amd64)
121         XTOI(px, hx);
122 #else
123         hx = px[0];
124 #endif
125         ix = hx & 0x7fffffff;

127 #endif /* ! codereview */
128         if (ix < 0x3ffc4000) {
129                 if (ix < 0x3fc60000)
130                         if ((i = (int)x) == 0)
131                                 return (one);

133                 /* generate inexact */
104                         if ((i = (int) x) == 0)
105                                 return (one);   /* generate inexact */
134                 z = x * x;

136                 if (ix < 0x3ff80000)    /* 0.0078125 */
137                         return (one + z * (qq1 + z * (qq2 + z * (qq3 + z *
138                             (qq4 + z * qq5)))));
139                 else
140                         return (one + z * (q1 + z * (q2 + z * (q3 + z * (q4 +
141                             z * (q5 + z * (q6 + z * (q7 + z * q8))))))));
142         }

144 #endif /* ! codereview */
145         j = (ix + 0x400) & 0x7ffff800;
146         i = (j - 0x3ffc4000) >> 11;
147 #if defined(__i386) || defined(__amd64)
148         ITOX(j, pt);
149 #else
150         pt[0] = j;
151 #endif

153 #endif /* ! codereview */
154         if (hx > 0)
155                 x = y - (t - x);
156         else
157                 x = (-y) - (t + x);

159 #endif /* ! codereview */
160         a = _TBL_cosl_hi[i];
161         z = x * x;
162         t = z * (qq1 + z * (qq2 + z * (qq3 + z * (qq4 + z * qq5))));
163         w = x * (one + z * (pp1 + z * (pp2 + z * (pp3 + z * (pp4 + z * pp5)))));
115         w = x * (one + z * (pp1 + z * (pp2 + z * (pp3 + z * (pp4 + z *
116                 pp5)))));
164         t = _TBL_cosl_lo[i] - (_TBL_sinl_hi[i] * w - a * t);
165         return (a + t);
```



```
166 }
```

_____*unchanged_portion_omitted_*

```
**********************************************************
   14567 Thu Jun  6 20:38:07 2019
new/usr/src/lib/libm/common/LD/__lgammal.c
11210 libm should be cstyle(10NBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

/*
 * long double __k_lgammal(long double x, int *signgamlp);
 * K.C. Ng, August, 1989.
 *
 * We choose [1.5,2.5] to be the primary interval. Our algorithms
 * are mainly derived from
 *
 *
 *                                zeta(2)-1    2    zeta(3)-1    3
 * lgamma(2+s) = s*(1-euler) + --------- * s  -  --------- * s  + ...
 *                                     2                 3
 *
 *
 * Note 1. Since gamma(1+s)=s*gamma(s), hence
 *              lgamma(1+s) = log(s) + lgamma(s), or
 *              lgamma(s) = lgamma(1+s) - log(s).
 *         When s is really tiny (like roundoff), lgamma(1+s) ~ s(1-enler)
 *         Hence lgamma(s) ~ -log(s) for tiny s
 *
 */

#include "libm.h"
#include "longdouble.h"

static long double neg(long double, int *);
static long double poly(long double, const long double *, int);
static long double polytail(long double);
static long double primary(long double);

static const long double c0 = 0.0L,
```



```
        ch = 0.5L,
        c1 = 1.0L,
        c2 = 2.0L,
        c3 = 3.0L,
        c4 = 4.0L,
        c5 = 5.0L,
        c6 = 6.0L,
        pi = 3.14159265358979323846264338327950288419 71L,
        tiny = 1.0e-40L;
static const long double
c0 =     0.0L,
ch =     0.5L,
c1 =     1.0L,
c2 =     2.0L,
c3 =     3.0L,
c4 =     4.0L,
c5 =     5.0L,
c6 =     6.0L,
pi =     3.14159265358979323846264338327950288419 71L,
tiny =   1.0e-40L;

long double
__k_lgammal(long double x, int *signgamlp)
{
__k_lgammal(long double x, int *signgamlp) {
        long double t, y;
        int i;

        /* purge off +-inf, NaN and negative arguments */
        if (!finitel(x))
                return (x * x);

                return (x*x);
        *signgamlp = 1;

#endif /* ! codereview */
        if (signbitl(x))
                return (neg(x, signgamlp));

        /* for x < 8.0 */
        if (x < 8.0L) {
                y = anintl(x);
                i = (int)y;

            i = (int) y;
                switch (i) {
                case 0:

#endif /* ! codereview */
                        if (x < 1.0e-40L)
                                return (-logl(x));
                        else
                                return ((primary(x) - log1pl(x)) - logl(x));

                        return (primary(x)-log1pl(x))-logl(x);
                case 1:
                        return (primary(x - y) - logl(x));
                return (primary(x-y)-logl(x));
                case 2:
                        return (primary(x - y));
                return (primary(x-y));
                case 3:
                        return (primary(x - y) + logl(x - c1));
                return (primary(x-y)+logl(x-c1));
                case 4:
                        return (primary(x - y) + logl((x - c1) * (x - c2)));
```

```
                return (primary(x-y)+logl((x-c1)*(x-c2)));
                case 5:
                        return (primary(x - y) + logl((x - c1) * (x - c2) * (x -
                            c3)));
                return (primary(x-y)+logl((x-c1)*(x-c2)*(x-c3)));
                case 6:
                        return (primary(x - y) + logl((x - c1) * (x - c2) * (x -
                            c3) * (x - c4)));
                return (primary(x-y)+logl((x-c1)*(x-c2)*(x-c3)*(x-c4)));
                case 7:
                        return (primary(x - y) + logl((x - c1) * (x - c2) * (x -
                            c3) * (x - c4) * (x - c5)));
                return (primary(x-y)+logl((x-c1)*(x-c2)*(x-c3)*(x-c4)*(x-c5)));
                case 8:
                        return (primary(x - y) + logl((x - c1) * (x - c2) * (x -
                            c3) * (x - c4) * (x - c5) * (x - c6)));
                return primary(x-y)+
                        logl((x-c1)*(x-c2)*(x-c3)*(x-c4)*(x-c5)*(x-c6));
                }
        }

        /* 8.0 <= x < 1.0e40 */
        if (x < 1.0e40L) {
                t = logl(x);
                return (x * (t - c1) - (ch * t - polytail(c1 / x)));
            return (x*(t-c1)-(ch*t-polytail(c1/x)));
        }

        /* 1.0e40 <= x <= inf */
        return (x * (logl(x) - c1));
        return (x*(logl(x)-c1));
}
```

_____*unchanged_portion_omitted_*

```
static long double
primary(long double s)                  /* assume |s|<=0.5 */
{
primary(long double s) {        /* assume |s|<=0.5 */
        int i;

        i = (int)(8.0L * (s + 0.5L));

        i = (int) (8.0L * (s + 0.5L));
        switch (i) {
        case 0:
                return (ch * s + s * poly(s, an4, 21));
        case 1:
                return (ch * s + s * poly(s, an3, 20));
        case 2:
                return (ch * s + s * poly(s, an2, 20));
        case 3:
                return (ch * s + s * poly(s, an1, 20));
        case 4:
                return (ch * s + s * poly(s, ap1, 19));
        case 5:
                return (ch * s + s * poly(s, ap2, 19));
        case 6:
                return (ch * s + s * poly(s, ap3, 19));
        case 7:
                return (ch * s + s * poly(s, ap4, 19));
        case 0: return ch*s+s*poly(s, an4, 21);
        case 1: return ch*s+s*poly(s, an3, 20);
        case 2: return ch*s+s*poly(s, an2, 20);
        case 3: return ch*s+s*poly(s, an1, 20);
        case 4: return ch*s+s*poly(s, ap1, 19);
        case 5: return ch*s+s*poly(s, ap2, 19);
```



```
        case 6: return ch*s+s*poly(s, ap3, 19);
        case 7: return ch*s+s*poly(s, ap4, 19);
        }

#endif /* ! codereview */
        /* NOTREACHED */
        return (0.0L);
}

static long double
poly(long double s, const long double *p, int n)
{
poly(long double s, const long double *p, int n) {
        long double y;
        int i;

        y = p[n - 1];

        for (i = n - 2; i >= 0; i--)
                y = p[i] + s * y;

        y = p[n-1];
        for (i = n-2; i >= 0; i--) y = p[i]+s*y;
        return (y);
}
```

_____*unchanged_portion_omitted_*

```
static long double
polytail(long double s)
{
polytail(long double s) {
        long double t, z;
        int i;

        z = s * s;
        z = s*s;
        t = pt[18];

        for (i = 17; i >= 1; i--)
                t = pt[i] + z * t;

        return (pt[0] + s * t);
        for (i = 17; i >= 1; i--) t = pt[i]+z*t;
        return (pt[0]+s*t);
}

static long double
neg(long double z, int *signgamlp)
{
neg(long double z, int *signgamlp) {
        long double t, p;

        /* BEGIN CSTYLED */
#endif /* ! codereview */
        /*
         * written by K.C. Ng,  Feb 2, 1989.
         *
         * Since
         *               -z*G(-z)*G(z) = pi/sin(pi*z),
         * we have
         *      G(-z) = -pi/(sin(pi*z)*G(z)*z)
         *                =  pi/(sin(pi*(-z))*G(z)*z)
         * Algorithm
         *               z = |z|
         *               t = sinpi(z); ...note that when z>2**112, z is an int
         *               and hence t=0.
```

```
 417          *
 418          *              if (t == 0.0) return 1.0/0.0;
 419          *              if (t< 0.0) *signgamlp = -1; else t= -t;
 420          *              if (z<1.0e-40)  ...tiny z
 421          *                  return -log(z);
 422          *              else
 423          *                  return log(pi/(t*z))-lgamma(z);
 424          *
 425          */
 426         /* END CSTYLED */
 427 #endif /* ! codereview */

 429         t = sinpil(z);                  /* t := sin(pi*z) */

 431 #endif /* ! codereview */
 432         if (t == c0)                    /* return   1.0/0.0 =  +INF */
 433                 return (c1 / c0);
 320             return (c1/c0);

 435         z = -z;

 437 #endif /* ! codereview */
 438         if (z <= tiny)
 439                 p = -logl(z);
 440         else
 441                 p = logl(pi / (fabsl(t) * z)) - __k_lgammal(z, signgamlp);

 443         if (t < c0)
 444                 *signgamlp = -1;

 323                 p = logl(pi/(fabsl(t)*z)) - __k_lgammal(z, signgamlp);
 324         if (t < c0) *signgamlp = -1;
 446         return (p);
 447 }
```

_____*unchanged_portion_omitted_*

```
**********************************************************
    1190 Thu Jun  6 20:38:08 2019
new/usr/src/lib/libm/common/LD/__poly_libmq.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#include "libm.h"

long double
__poly_libmq(long double x, int n, long double p[])
long double __poly_libmq(x,n,p)
long double x,p[];
int n;
{
        long double t;
        int i;

        t = p[n - 1];

        for (i = n - 2; i >= 0; i--)
                t = p[i] + x * t;

        return (t);
        long double t; int i;
        t = p[n-1];
        for(i=n-2;i>=0;i--) t = p[i] + x*t;
        return t;
}
______unchanged_portion_omitted_
```

```
**********************************************************
    1939 Thu Jun  6 20:38:08 2019
new/usr/src/lib/libm/common/LD/__rem_pio2l.c
11210 libm should be cstyle(10NBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

/*
 * __rem_pio2l(x,y)
 *
 * return the remainder of x rem pi/2 in y[0]+y[1]
 * by calling __rem_pio2m
 */

#include "libm.h"
#include "longdouble.h"

extern const int _TBL_ipio2l_inf[];
static const long double two24l = 16777216.0L,

static const long double
    two24l = 16777216.0L,
        pio4 = 0.785398163397448309615660845819875721049 5L;

int
__rem_pio2l(long double x, long double *y)
{
        long double z, w;
        double t[3], v[5];
        int e0, i, nx, n, sign;

        sign = signbitl(x);
        z = fabsl(x);

#endif /* ! codereview */
        if (z <= pio4) {
                y[0] = x;
```



```
                y[1] = 0;
                return (0);
        }

#endif /* ! codereview */
        e0 = ilogbl(z) - 23;
        z = scalbnl(z, -e0);

#endif /* ! codereview */
        for (i = 0; i < 3; i++) {
                t[i] = (double)((int)(z));
                z = (z - (long double)t[i]) * two24l;
        }

#endif /* ! codereview */
        nx = 3;

        while (t[nx - 1] == 0.0)
        while (t[nx-1] == 0.0)
                nx--;                   /* omit trailing zeros */

#endif /* ! codereview */
        n = __rem_pio2m(t, v, e0, nx, 2, _TBL_ipio2l_inf);
        z = (long double)v[1];
        w = (long double)v[0];
        y[0] = z + w;
        y[1] = z - (y[0] - w);

#endif /* ! codereview */
        if (sign != 0) {
                y[0] = -y[0];
                y[1] = -y[1];
                return (-n);
        }

#endif /* ! codereview */
        return (n);
}
```

```
**********************************************************
    4910 Thu Jun  6 20:38:09 2019
new/usr/src/lib/libm/common/LD/__sincosl.c
11210 libm should be cstyle(10NBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */


/* INDENT OFF */
/*
 * long double __k_sincos( long double x, long double y, long double *c )
 * kernel sincosl function on [-pi/4, pi/4], pi/4 ~ 0.785398164
 * Input x is assumed to be bounded by ~pi/4 in magnitude.
 * Input y is the tail of x.
 * return sinl(x) with *c = cosl(x)
 *
 * Table look up algorithm
 *      see __k_sinl() and __k_cosl()
 */

#include "libm.h"

#include <sys/isa_defs.h>

extern const long double _TBL_sinl_hi[], _TBL_sinl_lo[], _TBL_cosl_hi[],
        _TBL_cosl_lo[];
static const long double one = 1.0;

static const long double
one     = 1.0,
/*
 * |sin(x) - (x+pp1*x^3+...+pp5*x^11)| <= 2^-122.32 for |x|<1/64
 */
static const long double
        pp1 = -1.6666666666666666666666666586782940810e-0001L,
        pp2 = 8.3333333333333333333330037236609293175 40e-0003L,
        pp3 = -1.9841269841269840760459034837783378 04470e-0004L,
```



```
        pp4 = 2.7557319223619066413197231062109009494 13e-0006L,
        pp5 = -2.5051983985709470190939984691350120576 73e-0008L;

pp1     = -1.6666666666666666666666666586782940810e-0001L,
pp2     =  8.3333333333333333333330037236609293175 40e-0003L,
pp3     = -1.9841269841269840760459034837783378 04470e-0004L,
pp4     =  2.7557319223619066413197231062109009494 13e-0006L,
pp5     = -2.5051983985709470190939984691350120576 73e-0008L,
/*
 * |(sin(x) - (x+p1*x^3+...+p8*x^17)|
 * |------------------------------- | <= 2^-116.17 for |x|<0.1953125
 * |                x               |
 */
static const long double
        p1 = -1.6666666666666666666666666666211262297e-0001L,
        p2 = 8.3333333333333333333333333301497876908541e-0003L,
        p3 = -1.9841269841269841269840413028811806219 22e-0004L,
        p4 = 2.7557319223985890641005873513072696210 93e-0006L,
        p5 = -2.5052108385441631293789069537655953938 73e-0008L,
        p6 = 1.6059043836432443750509982437785340742 73e-0010L,
        p7 = -7.6471627228006855169014561142708246226 99e-0013L,
        p8 = 2.8100464286619029617254288410688444626 03e-0015L;

p1      =  -1.6666666666666666666666666666211262297e-0001L,
p2      =   8.3333333333333333333333333301497876908541e-0003L,
p3      =  -1.9841269841269841269840413028811806219 22e-0004L,
p4      =   2.7557319223985890641005873513072696210 93e-0006L,
p5      =  -2.5052108385441631293789069537655953938 73e-0008L,
p6      =   1.6059043836432443750509982437785340742 73e-0010L,
p7      =  -7.6471627228006855169014561142708246226 99e-0013L,
p8      =   2.8100464286619029617254288410688444626 03e-0015L,
/*
 *
 * |cos(x) - (1+qq1*x^2+...+ qq5*x^10)| <= 2^-123.84 for |x|<=1/128
 */
static const long double
        qq1 = -4.9999999999999999999999999999999378373641e-0001L,
        qq2 = 4.1666666666666666666666654783993277031 30e-0002L,
        qq3 = -1.3888888888888888888058211230618051613 494e-0003L,
        qq4 = 2.4801587301561053777715856589053031118 66e-0005L,
        qq5 = -2.7557280997625263257364883766951570087 36e-0007L;

qq1     =  -4.9999999999999999999999999999999378373641e-0001L,
qq2     =   4.1666666666666666666666654783993277031 30e-0002L,
qq3     =  -1.3888888888888888888058211230618051613 494e-0003L,
qq4     =   2.4801587301561053777715856589053031118 66e-0005L,
qq5     =  -2.7557280997625263257364883766951570087 36e-0007L,
/*
 *
 * |cos(x) - (1+q1*x^2+...+ q8*x^16)| <= 2^-117.11 for |x|<= 0.15625
 */
static const long double
        q1 = -4.9999999999999999999999999999999756416975e-0001L,
        q2 = 4.1666666666666666666666666406006657725 8e-0002L,
        q3 = -1.3888888888888888888888777003639371696 37e-0003L,
        q4 = 2.4801587301587301584944684630318140835 59e-0005L,
        q5 = -2.7557319223985862763228192503560055428 71e-0007L,
        q6 = 2.0876756987674242614419597607298540178 55e-0009L,
        q7 = -1.1470744812396620890724521290107907747 61e-0011L,
        q8 = 4.7777616473996515997306634222635310347 82e-0014L;


q1      =  -4.9999999999999999999999999999999756416975e-0001L,
q2      =   4.1666666666666666666666666406006657725 8e-0002L,
q3      =  -1.3888888888888888888888777003639371696 37e-0003L,
q4      =   2.4801587301587301584944684630318140835 59e-0005L,
```

```
 83 q5      =  -2.755731922398586276322819250356005542871e-0007L,
 84 q6      =   2.087675698767424261441959760729854017855e-0009L,
 85 q7      =  -1.147074481239662089072452129010790774761e-0011L,
 86 q8      =   4.777761647399651599730663422263531034782e-0014L;
 87 /* INDENT ON */
103 long double
104 __k_sincosl(long double x, long double y, long double *c)
105 {
 89 __k_sincosl(long double x, long double y, long double *c) {
106         long double a1, a2, t, t1, t2, z, w;
107         int *pt = (int *)&t, *px = (int *)&x;
 91         int *pt = (int *) &t, *px = (int *) &x;
108         int i, j, hx, ix;

110         t = 1.0;
111 #if defined(__i386) || defined(__amd64)
112         XTOI(px, hx);
113 #else
114         hx = px[0];
115 #endif
116         ix = hx & 0x7fffffff;

118 #endif /* ! codereview */
119         if (ix < 0x3ffc4000) {
120                 if (ix < 0x3fc60000) {
121                         if (((int)x) == 0) {
101                 if (ix < 0x3fc60000)
102                         if (((int) x) == 0) {
122                                 *c = one;
123                                 return (x);
124                         }               /* generate inexact */
125                 }

127 #endif /* ! codereview */
128                 z = x * x;

130                 if (ix < 0x3ff80000) {
131                         *c = one + z * (qq1 + z * (qq2 + z * (qq3 + z * (qq4 +
132                             z * qq5))));
133                         t = z * (p1 + z * (p2 + z * (p3 + z * (p4 + z * (p5 +
134                             z * p6)))));
135                 } else {
136                         *c = one + z * (q1 + z * (q2 + z * (q3 + z * (q4 + z *
137                             (q5 + z * (q6 + z * (q7 + z * q8)))))));
138                         t = z * (p1 + z * (p2 + z * (p3 + z * (p4 + z * (p5 +
139                             z * (p6 + z * (p7 + z * p8)))))));
140                 }

142                 t = y + x * t;
143                 return (x + t);
144         }

146 #endif /* ! codereview */
147         j = (ix + 0x400) & 0x7ffff800;
148         i = (j - 0x3ffc4000) >> 11;
149 #if defined(__i386) || defined(__amd64)
150         ITOX(j, pt);
151 #else
152         pt[0] = j;
153 #endif

155 #endif /* ! codereview */
156         if (hx > 0)
157                 x = y - (t - x);
158         else
159                 x = (-y) - (t + x);
```



```
161 #endif /* ! codereview */
162         a1 = _TBL_sinl_hi[i];
163         z = x * x;
164         t = z * (qq1 + z * (qq2 + z * (qq3 + z * (qq4 + z * qq5))));
165         w = x * (one + z * (pp1 + z * (pp2 + z * (pp3 + z * (pp4 + z * pp5)))));
106         w = x * (one + z * (pp1 + z * (pp2 + z * (pp3 + z * (pp4 + z *
107                 pp5)))));
166         a2 = _TBL_cosl_hi[i];
167         t2 = _TBL_cosl_lo[i] - (a1 * w - a2 * t);
168         *c = a2 + t2;
169         t1 = a2 * w + a1 * t;
170         t1 += _TBL_sinl_lo[i];

172 #endif /* ! codereview */
173         if (hx < 0)
174                 return (-a1 - t1);
175         else
176                 return (a1 + t1);
177 }
```

```
**********************************************************
    4727 Thu Jun  6 20:38:10 2019
new/usr/src/lib/libm/common/LD/__sinl.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */


/* INDENT OFF */
/*
 * __k_sinl( long double x;  long double y )
 * kernel sin function on [-pi/4, pi/4], pi/4 ~ 0.785398164
 * Input x is assumed to be bounded by ~pi/4 in magnitude.
 * Input y is the tail of x.
 *
 * Table look up algorithm
 *      1. by sin(-x) = -sin(x), need only to consider positive x
 *      2. if x < 25/128 = [0x3ffc9000,0,0,0] = 0.1953125 , then
 *           if x < 2^-57 (hx < 0x3fc60000,0,0,0), return x (inexact if x !=  0)
 *           z = x*x;
 *           if x <= 1/64 = 2**-6
 *              sin(x) = x + (y+(x*z)*(p1 + z*p2))
 *           else
 *              sin(x) = x + (y+(x*z)*(p1 + z*(p2 + z*(p3 + z*p4))))
 *      3. else
 *              ht = (hx + 0x400)&0x7ffff800    (round x to a break point t)
 *              lt = 0
 *              i  = (hy-0x3ffc4000)>>11;       (i<=64)
 *              x' = (x - t)+y                  (|x'| ~<= 2^-7
 *         By
 *              sin(t+x')
 *                = sin(t)cos(x')+cos(t)sin(x')
 *                = sin(t)(1+z*(qq1+z*qq2))+[cos(t)]*x*(1+z*(pp1+z*pp2))
 *                = sin(t) + [sin(t)]*(z*(qq1+z*qq2))+
 *                              [cos(t)]*x*(1+z*(pp1+z*pp2))
 *
 *         Thus,
```



```
 *              let a= _TBL_sin_hi[i], b = _TBL_sin_lo[i], c= _TBL_cos_hi[i],
 *              x = (x-t)+y
 *              z = x*x;
 *              sin(t+x) = a+(b+ ((c*x)*(1+z*(pp1+z*pp2))+a*(z*(qq1+z*qq2)))
 */

#include "libm.h"

#include <sys/isa_defs.h>

extern const long double _TBL_sinl_hi[], _TBL_sinl_lo[], _TBL_cosl_hi[];
static const long double one = 1.0;

static const long double
one     = 1.0,
/*
 * |sin(x) - (x+pp1*x^3+...+ pp5*x^11)| <= 2^-122.32 for |x|<1/64
 */
static const long double
        pp1 = -1.6666666666666666666666666586782940810e-0001L,
        pp2 = 8.3333333333333333333300372366092931754 0e-0003L,
        pp3 = -1.9841269841269840760459034837783378047 0e-0004L,
        pp4 = 2.7557319223619066413197231062109009494 13e-0006L,
        pp5 = -2.5051983985709470190939984691350120576 73e-0008L;

pp1     = -1.6666666666666666666666666586782940810e-0001L,
pp2     =  8.3333333333333333333300372366092931754 0e-0003L,
pp3     = -1.9841269841269840760459034837783378047 0e-0004L,
pp4     =  2.7557319223619066413197231062109009494 13e-0006L,
pp5     = -2.5051983985709470190939984691350120576 73e-0008L,
/*
 * |(sin(x) - (x+p1*x^3+...+p8*x^17)|
 * |------------------------------- | <= 2^-116.17 for |x|<0.1953125
 * |                 x              |
 */
static const long double
        p1 = -1.6666666666666666666666666666211262297e-0001L,
        p2 = 8.3333333333333333333333333301497876908541e-0003L,
        p3 = -1.9841269841269841269840413028811806219 22e-0004L,
        p4 = 2.7557319223985890641005873513072696210 93e-0006L,
        p5 = -2.5052108385441631293789069537655953938 73e-0008L,
        p6 = 1.6059043836432443750509982437785340742 73e-0010L,
        p7 = -7.6471627228006855169014561142708246226 99e-0013L,
        p8 = 2.8100464286619029617254288410688444626 03e-0015L;

p1      = -1.6666666666666666666666666666211262297e-0001L,
p2      =  8.3333333333333333333333333301497876908541e-0003L,
p3      = -1.9841269841269841269840413028811806219 22e-0004L,
p4      =  2.7557319223985890641005873513072696210 93e-0006L,
p5      = -2.5052108385441631293789069537655953938 73e-0008L,
p6      =  1.6059043836432443750509982437785340742 73e-0010L,
p7      = -7.6471627228006855169014561142708246226 99e-0013L,
p8      =  2.8100464286619029617254288410688444626 03e-0015L,
/*
 *                  2           10        -123.84
 * |cos(x) - (1+qq1*x +...+ qq5*x  )| <= 2        for |x|<=1/128
 */
static const long double
        qq1 = -4.9999999999999999999999999999999378373641e-0001L,
        qq2 = 4.1666666666666666666666654783993277031 30e-0002L,
        qq3 = -1.3888888888888888888058211230618051613 494e-0003L,
        qq4 = 2.4801587301561053777715856589053031118 66e-0005L,
        qq5 = -2.7557280997625263257364883766951570087 36e-0007L;


qq1     = -4.9999999999999999999999999999999378373641e-0001L,
```

```
 93 qq2     =  4.16666666666666666666665478399327703130e-0002L,
 94 qq3     = -1.38888888888888888805821123061805161349 4e-0003L,
 95 qq4     =  2.48015873015610537777158565890530311186 6e-0005L,
 96 qq5     = -2.75572809976252632573648837669515700873 6e-0007L;
 97 /* INDENT ON */
111 long double
112 __k_sinl(long double x, long double y)
113 {
 99 __k_sinl(long double x, long double y) {
114         long double a, t, z, w;
115         int *pt = (int *)&t, *px = (int *)&x;
101         int *pt = (int *) &t, *px = (int *) &x;
116         int i, j, hx, ix;

118         t = 1.0L;
119 #if defined(__i386) || defined(__amd64)
120         XTOI(px, hx);
121 #else
122         hx = px[0];
123 #endif
124         ix = hx & 0x7fffffff;

126 #endif /* ! codereview */
127         if (ix < 0x3ffc9000) {
128                 if (ix < 0x3fc60000)
129                         if (((int)x) == 0)
130                                 return (x);

132                 /* generate inexact */
111                         if (((int) x) == 0)
112                                 return (x);     /* generate inexact */
133                 z = x * x;
134                 t = z * (p1 + z * (p2 + z * (p3 + z * (p4 + z * (p5 + z * (p6 +
135                     z * (p7 + z * p8)))))));
114                 t = z * (p1 + z * (p2 + z * (p3 + z * (p4 + z * (p5 + z *
115                         (p6 + z * (p7 + z * p8)))))));
136                 t = y + x * t;
137                 return (x + t);
138         }

140 #endif /* ! codereview */
141         j = (ix + 0x400) & 0x7ffff800;
142         i = (j - 0x3ffc4000) >> 11;
143 #if defined(__i386) || defined(__amd64)
144         ITOX(j, pt);
145 #else
146         pt[0] = j;
147 #endif

149 #endif /* ! codereview */
150         if (hx > 0)
151                 x = y - (t - x);
152         else
153                 x = (-y) - (t + x);

155 #endif /* ! codereview */
156         a = _TBL_sinl_hi[i];
157         z = x * x;
158         t = z * (qq1 + z * (qq2 + z * (qq3 + z * (qq4 + z * qq5))));
159         w = x * (one + z * (pp1 + z * (pp2 + z * (pp3 + z * (pp4 + z * pp5)))));
119         w = x * (one + z * (pp1 + z * (pp2 + z * (pp3 + z * (pp4 + z *
120                 pp5)))));
160         t = _TBL_cosl_hi[i] * w + a * t;
161         t += _TBL_sinl_lo[i];

163 #endif /* ! codereview */
```



```
164         if (hx < 0)
165                 return (-a - t);
166         else
167                 return (a + t);
168 }
```

```
**********************************************************
    5572 Thu Jun  6 20:38:10 2019
new/usr/src/lib/libm/common/LD/__tanl.c
11210 libm should be cstyle(10NBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */


/* INDENT OFF */
/*
 * __k_tanl( long double x;  long double y; int k )
 * kernel tan/cotan function on [-pi/4, pi/4], pi/4 ~ 0.785398164
 * Input x is assumed to be bounded by ~pi/4 in magnitude.
 * Input y is the tail of x.
 * Input k indicate -- tan if k=0; else -1/tan
 *
 * Table look up algorithm
 *      1. by tan(-x) = -tan(x), need only to consider positive x
 *      2. if x < 5/32 = [0x3ffc4000, 0] = 0.15625 , then
 *           if x < 2^-57 (hx < 0x3fc40000 0), set w=x with inexact if x !=  0
 *           else
 *              z = x*x;
 *              w = x + (y+(x*z)*(t1+z*(t2+z*(t3+z*(t4+z*(t5+z*t6))))))
 *         return (k == 0 ? w : 1/w);
 *      3. else
 *              ht = (hx + 0x400)&0x7ffff800    (round x to a break point t)
 *              lt = 0
 *              i  = (hy-0x3ffc4000)>>11;       (i<=64)
 *              x' = (x - t)+y                  (|x'| ~<= 2^-7)
 *         By
 *              tan(t+x')
 *                = (tan(t)+tan(x'))/(1-tan(x')tan(t))
 *         We have
 *                              sin(x')+tan(t)*(tan(t)*sin(x'))
 *                = tan(t) + ------------------------------    for k=0
 *                                  cos(x') - tan(t)*sin(x')
```



```
 *                              cos(x') - tan(t)*sin(x')
 *                = - --------------------------------------    for k=1
 *                      tan(t) + tan(t)*(cos(x')-1) + sin(x')
 *
 *
 *         where        tan(t) is from the table,
 *                      sin(x') = x + pp1*x^3 + ...+ pp5*x^11
 *                      cos(x') = 1 + qq1*x^2 + ...+ qq5*x^10
 */

#include "libm.h"

#include <sys/isa_defs.h>

extern const long double _TBL_tanl_hi[], _TBL_tanl_lo[];
static const long double one = 1.0;

static const long double
one     = 1.0,
/*
 * |sin(x) - (x+pp1*x^3+...+ pp5*x^11)| <= 2^-122.32 for |x|<1/64
 */
static const long double
        pp1 = -1.6666666666666666666666666666586782940810e-0001L,
        pp2 = 8.3333333333333333333330037236609293175 40e-0003L,
        pp3 = -1.9841269841269840760459034837783378044 70e-0004L,
        pp4 = 2.7557319223619066413197231062109009494 13e-0006L,
        pp5 = -2.5051983985709470190939984691350120576 73e-0008L;

pp1     = -1.6666666666666666666666666666586782940810e-0001L,
pp2     =  8.3333333333333333333330037236609293175 40e-0003L,
pp3     = -1.9841269841269840760459034837783378044 70e-0004L,
pp4     =  2.7557319223619066413197231062109009494 13e-0006L,
pp5     = -2.5051983985709470190939984691350120576 73e-0008L,
/*
 *                    2           10         -123.84
 * |cos(x) - (1+qq1*x +...+ qq5*x  )| <= 2        for |x|<=1/128
 */
static const long double
        qq1 = -4.9999999999999999999999999999999378373641e-0001L,
        qq2 = 4.1666666666666666666666547839932770313 0e-0002L,
        qq3 = -1.3888888888888888880582112306180516134 94e-0003L,
        qq4 = 2.4801587301561053777715856589053031118 66e-0005L,
        qq5 = -2.7557280997625263257364883766957157008 736e-0007L;

qq1     = -4.9999999999999999999999999999999378373641e-0001L,
qq2     =  4.1666666666666666666666547839932770313 0e-0002L,
qq3     = -1.3888888888888888880582112306180516134 94e-0003L,
qq4     =  2.4801587301561053777715856589053031118 66e-0005L,
qq5     = -2.7557280997625263257364883766957157008 736e-0007L,
/*
 * |tan(x) - (x+t1*x^3+...+t6*x^13)|
 * |------------------------------- | <= 2^-59.73 for |x|<0.15625
 * |                x               |
 */
static const long double
        t1 = 3.3333333333333333333333333333333423342490e-0001L,
        t2 = 1.3333333333333333333333333333093838744537e-0001L,
        t3 = 5.3968253968253968253968279063186826622 50e-0002L,
        t4 = 2.1869488536155202821855769769944184869 11e-0002L,
        t5 = 8.8632355299021965733545451999115293624 6e-0003L,
        t6 = 3.5921280365724800646521914275439948787 90e-0003L,
        t7 = 1.4558343870514552578568338075819013054 74e-0003L,
        t8 = 5.9002744093185998578299832562017255874 77e-0004L,
        t9 = 2.3912911521172651815011169619011223629 37e-0004L,
        t10 = 9.6915331693827297423940241731949818823 75e-0005L,
```

```
115         t11 = 3.927994733186415603228178184225780859951e-0005L,
116         t12 = 1.588300018848323824227640064883334101288e-0005L,
117         t13 = 6.916271223396808311166202285131722231723e-0006L;


 93 t1      =  3.333333333333333333333333333333423342490e-0001L,
 94 t2      =  1.333333333333333333333333333309838744537e-0001L,
 95 t3      =  5.396825396825396825396827906318682662250e-0002L,
 96 t4      =  2.186948853615520282185576976994418486911e-0002L,
 97 t5      =  8.863235529902196573354554519991152936246e-0003L,
 98 t6      =  3.592128036572480064652191427543994878790e-0003L,
 99 t7      =  1.455834387051455257856833807581901305474e-0003L,
100 t8      =  5.900274409318599857829983256201725587477e-0004L,
101 t9      =  2.391291152117265181501116961901122362937e-0004L,
102 t10     =  9.691533169382729742394024173194981882375e-0005L,
103 t11     =  3.927994733186415603228178184225780859951e-0005L,
104 t12     =  1.588300018848323824227640064883334101288e-0005L,
105 t13     =  6.916271223396808311166202285131722231723e-0006L;
106 /* INDENT ON */
120 long double
121 __k_tanl(long double x, long double y, int k)
122 {
108 __k_tanl(long double x, long double y, int k) {
123         long double a, t, z, w = 0.0, s, c;
124         int *pt = (int *)&t, *px = (int *)&x;
110         int *pt = (int *) &t, *px = (int *) &x;
125         int i, j, hx, ix;

127         t = 1.0;
128 #if defined(__i386) || defined(__amd64)
129         XTOI(px, hx);
130 #else
131         hx = px[0];
132 #endif
133         ix = hx & 0x7fffffff;

135 #endif /* ! codereview */
136         if (ix < 0x3ffc4000) {
137                 if (ix < 0x3fc60000) {
138                         if ((i = (int)x) == 0)  /* generate inexact */
120                         if ((i = (int) x) == 0) /* generate inexact */
139                                 w = x;
140                 } else {
141                         z = x * x;

143                         if (ix < 0x3ff30000) {  /* 2**-12 */
124                         if (ix < 0x3ff30000)    /* 2**-12 */
144                                 t = z * (t1 + z * (t2 + z * (t3 + z * t4)));
145                         } else {
146                                 t = z * (t1 + z * (t2 + z * (t3 + z * (t4 + z *
147                                     (t5 + z * (t6 + z * (t7 + z * (t8 + z *
148                                     (t9 + z * (t10 + z * (t11 + z * (t12 + z *
149                                     t13)))))))))))));
150                         }

126                         else
127                                 t = z * (t1 + z * (t2 + z * (t3 + z * (t4 +
128                                         z * (t5 + z * (t6 + z * (t7 + z *
129                                         (t8 + z * (t9 + z * (t10 + z * (t11 +
130                                         z * (t12 + z * t13)))))))))))));
152                         t = y + x * t;
153                         w = x + t;
154                 }

156 #endif /* ! codereview */
157                 return (k == 0 ? w : -one / w);
```



```
158         }

160 #endif /* ! codereview */
161         j = (ix + 0x400) & 0x7ffff800;
162         i = (j - 0x3ffc4000) >> 11;
163 #if defined(__i386) || defined(__amd64)
164         ITOX(j, pt);
165 #else
166         pt[0] = j;
167 #endif

169 #endif /* ! codereview */
170         if (hx > 0)
171                 x = y - (t - x);
172         else
173                 x = (-y) - (t + x);

175 #endif /* ! codereview */
176         a = _TBL_tanl_hi[i];
177         z = x * x;
178         /* cos(x)-1 */
179         t = z * (qq1 + z * (qq2 + z * (qq3 + z * (qq4 + z * qq5))));
180         /* sin(x) */
181         s = x * (one + z * (pp1 + z * (pp2 + z * (pp3 + z * (pp4 + z * pp5)))));

134         s = x * (one + z * (pp1 + z * (pp2 + z * (pp3 + z * (pp4 + z *
135                 pp5)))));
183         if (k == 0) {
184                 w = a * s;
185                 t = _TBL_tanl_lo[i] + (s + a * w) / (one - (w - t));
186                 return (hx < 0 ? -a - t : a + t);
187         } else {
188                 w = s + a * t;
189                 c = w + _TBL_tanl_lo[i];
190                 z = (one - (a * s - t));
191                 return (hx >= 0 ? z / (-a - c) : z / (a + c));
192         }
193 }
```

_____*unchanged_portion_omitted_*

```
**********************************************************
    1488 Thu Jun  6 20:38:11 2019
new/usr/src/lib/libm/common/LD/acoshl.c
11210 libm should be cstyle(10NBLD) clean
**********************************************************
   1 /*
   2  * CDDL HEADER START
   3  *
   4  * The contents of this file are subject to the terms of the
   5  * Common Development and Distribution License (the "License").
   6  * You may not use this file except in compliance with the License.
   7  *
   8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
   9  * or http://www.opensolaris.org/os/licensing.
  10  * See the License for the specific language governing permissions
  11  * and limitations under the License.
  12  *
  13  * When distributing Covered Code, include this CDDL HEADER in each
  14  * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  15  * If applicable, add the following below this CDDL HEADER, with the
  16  * fields enclosed by brackets "[]" replaced with your own identifying
  17  * information: Portions Copyright [yyyy] [name of copyright owner]
  18  *
  19  * CDDL HEADER END
  20  */

  22 /*
  23  * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
  24  */

  26 #endif /* ! codereview */
  27 /*
  28  * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
  29  * Use is subject to license terms.
  30  */

  32 #pragma weak __acoshl = acoshl

  34 #include "libm.h"

  36 static const long double zero = 0.0L,
  25 static const long double
  26         zero    = 0.0L,
  37         ln2 = 6.931471805599453094172321214581765680755e-0001L,
  38         one = 1.0L,
  39         big = 1.e+20L;

  41 long double
  42 acoshl(long double x)
  43 {
  32 acoshl(long double x) {
  44         long double t;

  46         if (isnanl(x)) {
  35         if (isnanl(x))
  47                 return (x + x);
  48         } else if (x > big) {
  37         else if (x > big)
  49                 return (logl(x) + ln2);
  50         } else if (x > one) {
  39         else if (x > one) {
  51                 t = sqrtl(x - one);
  52                 return (log1pl(t * (t + sqrtl(x + one))));
  53         } else if (x == one) {
  42         } else if (x == one)
  54                 return (zero);
```



```
  55         } else {
  44         else
  56                 return ((x - x) / (x - x));
  57         }
  58 #endif /* ! codereview */
  59 }
```

```
**********************************************************
    1630 Thu Jun  6 20:38:11 2019
new/usr/src/lib/libm/common/LD/asinhl.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __asinhl = asinhl

#include "libm.h"

static const long double
        ln2 = 6.931471805599453094172321214581765680755e-0001L,
        one = 1.0L,
        big = 1.0e+20L,
        tiny = 1.0e-20L;

long double
asinhl(long double x)
{
asinhl(long double x) {
        long double t, w;

#endif /* ! codereview */
#ifndef lint
        volatile long double dummy __unused;
#endif

        w = fabsl(x);

#endif /* ! codereview */
        if (isnanl(x))
                return (x + x);         /* x is NaN */

#endif /* ! codereview */
        if (w < tiny) {
#ifndef lint
```



```
                dummy = x + big;        /* inexact if x != 0 */
#endif
                return (x);             /* tiny x */
        } else if (w < big) {
                t = one / w;
                return (copysignl(log1pl(w + w / (t + sqrtl(one + t * t))), x));
        } else {
        } else
                return (copysignl(logl(w) + ln2, x));
        }
#endif /* ! codereview */
}
```

```
**********************************************************
    1340 Thu Jun  6 20:38:11 2019
new/usr/src/lib/libm/common/LD/atan2pil.c
11210 libm should be cstyle(10NBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#include "libm.h"

#define GENERIC         long double
#define ATAN2PI         atan2pil
#define ATAN2           atan2l

/*
 * ATAN2PI(y, x)
/* ATAN2PI(y,x)
 *
 *      ATAN2PI(y, x) = ATAN2(y, x) / pi
 *      ATAN2PI(y,x) = ATAN2(y,x)/pi
 */

extern GENERIC ATAN2();
static GENERIC invpi =
        (GENERIC)3.183098861837906715377675267450287240689e-0001L;
#endif /* ! codereview */

GENERIC
ATAN2PI(GENERIC y, GENERIC x)
static GENERIC
invpi = (GENERIC) 3.183098861837906715377675267450287240689e-0001L;

GENERIC ATAN2PI(y,x)
GENERIC y,x;
{
        return (ATAN2(y, x) * invpi);
        return ATAN2(y,x)*invpi;
}
```

```
**********************************************************
    2034 Thu Jun  6 20:38:12 2019
new/usr/src/lib/libm/common/LD/atanhl.c
11210 libm should be cstyle(10NBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __atanhl = atanhl

#include "libm.h"

#define GENERIC         long double
#define ATANH           atanhl

/* BEGIN CSTYLED */
/*
 * ATANH(x)
 * ATANH(x)
                    1              2x                          x
      ATANH(x) = --- * LOG(1 + -------) = 0.5 * LOG1P(2 * --------)
                    2            1 - x                      1 - x
 * Note: to guarantee ATANH(-x) = -ATANH(x), we use
                  sign(x)             |x|
      ATANH(x) = ------- * LOG1P(2*-------).
                     2              1 - |x|
 *
 * Special cases:
      ATANH(x) is NaN if |x| > 1 with signal;
      ATANH(NaN) is that NaN with no signal;
      ATANH(+-1) is +-INF with signal.
 *
 */
/* END CSTYLED */
#endif /* ! codereview */

#define FABS            fabsl
#define LOG1P           log1pl
```



```
#define COPYSIGN        copysignl

extern GENERIC FABS(), LOG1P(), COPYSIGN();
#endif /* ! codereview */

static GENERIC zero = (GENERIC)0.0, half = (GENERIC)0.5, one = (GENERIC)1.0;
extern GENERIC  FABS(),LOG1P(),COPYSIGN();

static GENERIC
zero    = (GENERIC) 0.0,
half    = (GENERIC) 0.5,
one     = (GENERIC) 1.0;

GENERIC
ATANH(GENERIC x)
GENERIC ATANH(x)
GENERIC x;
{
        GENERIC t;

#endif /* ! codereview */
        t = FABS(x);

        if (t == one)
                return (x / zero);

        t = t / (one - t);
        return (COPYSIGN(half, x) * LOG1P(t + t));
        if (t == one) return x/zero;
        t = t/(one-t);
        return COPYSIGN(half,x)*LOG1P(t+t);
}
```

_____unchanged_portion_omitted_

```
**********************************************************
    1753 Thu Jun  6 20:38:12 2019
new/usr/src/lib/libm/common/LD/cbrtl.c
11210 libm should be cstyle(10NBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __cbrtl = cbrtl

#include "libm.h"
#include "longdouble.h"

static const double d_one = 1.0;

long double
cbrtl(long double x)
{
cbrtl(long double x) {
        long double s, t, r, w, y;
        double dx, dy;
        int *py = (int *)&dy;
        int *py = (int *) &dy;
        int n, m, m3, n0, sx;

        if (!finitel(x))
                return (x + x);

#endif /* ! codereview */
        if (iszerol(x))
                return (x);

#endif /* ! codereview */
        n0 = 0;

        if (*((int *)&d_one) == 0)
        if (*((int *) &d_one) == 0)
                n0 = 1;
```



```
#endif /* ! codereview */
        sx = signbitl(x);
        x = fabsl(x);
        n = ilogbl(x);
        m = n / 3;
        m3 = m + m + m;
        y = scalbnl(x, -m3);
        dx = (double)y;
        dx = (double) y;
        dy = cbrt(dx);
        py[1 - n0] += 2;

#endif /* ! codereview */
        if (py[1 - n0] == 0)
                py[n0] += 1;

        /* one step newton iteration to 113 bits with error < 0.667ulps */
        t = (long double)dy;
        t = (long double) dy;
        t = scalbnl(t, m);
        s = t * t;
        r = x / s;
        w = t + t;
        r = (r - t) / (w + r);
        t += t * r;

        return (sx == 0 ? t : -t);
}
______unchanged_portion_omitted_
```

```
**********************************************************
    2802 Thu Jun  6 20:38:13 2019
new/usr/src/lib/libm/common/LD/coshl.c
11210 libm should be cstyle(10NBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __coshl = coshl

#include "libm.h"
#include "longdouble.h"

/*
 * COSH(X)
 * RETURN THE HYPERBOLIC COSINE OF X
 *
 * Method :
 *      1. Replace x by |x| (COSH(x) = COSH(-x)).
 *      2.
 *                                                      [ EXP(x) - 1 ]^2
 *          0        <= x <= 0.3465  :  COSH(x) := 1 + -------------------
 *                                                         2*EXP(x)
 *
 *                                                 EXP(x) +  1/EXP(x)
 *          0.3465   <= x <= thresh  :  COSH(x) := -------------------
 *                                                          2
 *          thresh   <= x <= lnovft  :  COSH(x) := EXP(x)/2
 *          lnovft   <= x <  INF     :  COSH(x) := SCALBN(EXP(x-MEP1*ln2),ME)
 *
 *
 * here
 *      0.3465          a number that is near one half of ln2.
 *      thresh          a number such that
 *                              EXP(thresh)+EXP(-thresh)=EXP(thresh)
 *      lnovft          logarithm of the overflow threshold
 *                      = MEP1*ln2 chopped to machine precision.
 *      ME              maximum exponent
```



```
 *      MEP1            maximum exponent plus 1
 *
 * Special cases:
 *      COSH(x) is |x| if x is +INF, -INF, or NaN.
 *      only COSH(0)=1 is exact for finite x.
 */

static const long double C[] = {
        0.5L,
        1.0L,
        0.3465L,
        45.0L,
        1.13565234062941439487914 9e+04L,
        7.00444768624254908785898 5e-16L,
        2.71050543121376108501863 2e-20L, /* 2^-65 */
};

#define half            C[0]
#define one             C[1]
#define thr1            C[2]
#define thr2            C[3]
#define lnovft          C[4]
#define lnovlo          C[5]
#define tinyl           C[6]

long double
coshl(long double x)
{
coshl(long double x) {
        long double w, t;

        w = fabsl(x);

#endif /* ! codereview */
        if (!finitel(w))
                return (w + w);         /* x is INF or NaN */

#endif /* ! codereview */
        if (w < thr1) {
                if (w < tinyl)
                        return (one + w);       /* inexact+directed rounding */

#endif /* ! codereview */
                t = expm1l(w);
                w = one + t;
                w = one + (t * t) / (w + w);
                return (w);
        }

#endif /* ! codereview */
        if (w < thr2) {
                t = expl(w);
                return (half * (t + one / t));
        }

#endif /* ! codereview */
        if (w <= lnovft)
                return (half * expl(w));

#endif /* ! codereview */
        return (scalbnl(expl((w - lnovft) - lnovlo), 16383));
}
```

```
**********************************************************
    2816 Thu Jun  6 20:38:13 2019
new/usr/src/lib/libm/common/LD/cosl.c
11210 libm should be cstyle(10NBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __cosl = cosl

/* BEGIN CSTYLED */
/*
 * cosl(x)
/* INDENT OFF */
/* cosl(x)
 * Table look-up algorithm by K.C. Ng, November, 1989.
 *
 * kernel function:
 *      __k_sinl        ... sin function on [-pi/4,pi/4]
 *      __k_cosl        ... cos function on [-pi/4,pi/4]
 *      __rem_pio2l     ... argument reduction routine
 *
 * Method.
 *      Let S and C denote the sin and cos respectively on [-PI/4, +PI/4].
 *      1. Assume the argument x is reduced to y1+y2 = x-k*pi/2 in
 *         [-pi/2 , +pi/2], and let n = k mod 4.
 *      2. Let S=S(y1+y2), C=C(y1+y2). Depending on n, we have
 *
 *          n        sin(x)      cos(x)        tan(x)
 *     ----------------------------------------------------------
 *          0          S           C             S/C
 *          1          C          -S            -C/S
 *          2         -S          -C             S/C
 *          3         -C           S            -C/S
 *     ----------------------------------------------------------
 *
 * Special cases:
 *      Let trig be any of sin, cos, or tan.
```



```
 *      trig(+-INF)  is NaN, with signals;
 *      trig(NaN)    is that NaN;
 *
 * Accuracy:
 *      computer TRIG(x) returns trig(x) nearly rounded.
 */
/* END CSTYLED */
/* INDENT ON */

#include "libm.h"
#include "longdouble.h"

#include <sys/isa_defs.h>

long double
cosl(long double x)
{
cosl(long double x) {
        long double y[2], z = 0.0L;
        int n, ix;
        int *px = (int *)&x;
        int *px = (int *) &x;

        /* trig(Inf or NaN) is NaN */
        if (!finitel(x))
                return (x - x);
                return x - x;

        /* High word of x. */
#if defined(__i386) || defined(__amd64)
        XTOI(px, ix);
#else
        ix = px[0];
#endif

        /* |x| ~< pi/4 */
        ix &= 0x7fffffff;
        if (ix <= 0x3ffe9220)
                return __k_cosl(x, z);

        if (ix <= 0x3ffe9220) {
                return (__k_cosl(x, z));
        }
#endif /* ! codereview */
        /* argument reduction needed */
        else {
                n = __rem_pio2l(x, y);

#endif /* ! codereview */
                switch (n & 3) {
                case 0:
                        return (__k_cosl(y[0], y[1]));
                        return __k_cosl(y[0], y[1]);
                case 1:
                        return (-__k_sinl(y[0], y[1]));
                        return -__k_sinl(y[0], y[1]);
                case 2:
                        return (-__k_cosl(y[0], y[1]));
                        return -__k_cosl(y[0], y[1]);
                case 3:
                        return (__k_sinl(y[0], y[1]));
                        return __k_sinl(y[0], y[1]);
                        /* NOTREACHED */
                }
        }
```

```
116         return (0.0L);
 95     return 0.0L;
117 }
```

_____*unchanged_portion_omitted_*

```
**********************************************************
   12767 Thu Jun  6 20:38:14 2019
new/usr/src/lib/libm/common/LD/erfl.c
11210 libm should be cstyle(10NBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

/* BEGIN CSTYLED */
/*
 * long double function erf,erfc (long double x)
/* long double function erf,erfc (long double x)
 * K.C. Ng, September, 1989.
 *                       x
 *                 2     |\
 *     erf(x)  =  ---------  | exp(-t*t)dt
 *                sqrt(pi) \|
 *                         0
 *
 *     erfc(x) =  1-erf(x)
 *
 * method:
 *      Since erf(-x) = -erf(x), we assume x>=0.
 *      For x near 0, we have the expansion
 *
 *          erf(x) = (2/sqrt(pi))*(x - x^3/3 + x^5/10 - x^7/42 + ....).
 *
 *      Since 2/sqrt(pi) = 1.128379167095512573896158903121545171688,
 *      we use x + x*P(x^2) to approximate erf(x). This formula will
 *      guarantee the error less than one ulp where x is not too far
 *      away from 0. We note that erf(x)=x at x = 0.6174..... After
 *      some experiment, we choose the following approximation on
 *      interval [0,0.84375].
 *
 *      For x in [0,0.84375]
 *                 2                 2        4                40
 *         P =  P(x ) = (p0 + p1 * x + p2 * x + ... + p20 * x  )
 *
```



```
 *         erf(x)  = x + x*P
 *         erfc(x) = 1 - erf(x)           if x<=0.25
 *                 = 0.5 + ((0.5-x)-x*P)  if x in [0.25,0.84375]
 *      precision: |P(x^2)-(erf(x)-x)/x| <= 2**-122.50
 *
 *      For x in [0.84375,1.25], let s = x - 1, and
 *      c = 0.84506291151 rounded to single (24 bits)
 *         erf(x)  = c  + P1(s)/Q1(s)
 *         erfc(x) = (1-c)  - P1(s)/Q1(s)
 *      precision: |P1/Q1 - (erf(x)-c)| <= 2**-118.41
 *
 *
 *      For x in [1.25,1.75], let s = x - 1.5, and
 *      c = 0.95478588343 rounded to single (24 bits)
 *         erf(x)  = c  + P2(s)/Q2(s)
 *         erfc(x) = (1-c)  - P2(s)/Q2(s)
 *      precision: |P1/Q1 - (erf(x)-c)| <= 2**-123.83
 *
 *
 *      For x in [1.75,16/3]
 *         erfc(x) = exp(-x*x)*(1/x)*R1(1/x)/S1(1/x)
 *         erf(x)  = 1 - erfc(x)
 *      precision: absolute error of R1/S1 is bounded by 2**-124.03
 *
 *      For x in [16/3,107]
 *         erfc(x) = exp(-x*x)*(1/x)*R2(1/x)/S2(1/x)
 *         erf(x)  = 1 - erfc(x) (if x>=9 simple return erf(x)=1 with inexact)
 *      precision: absolute error of R2/S2 is bounded by 2**-120.07
 *
 *      Else if inf > x >= 107
 *         erf(x)  = 1 with inexact
 *         erfc(x) = 0 with underflow
 *
 *      Special case:
 *         erf(inf)  = 1
 *         erfc(inf) = 0
 */
/* END CSTYLED */
#endif /* ! codereview */

#pragma weak __erfl = erfl
#pragma weak __erfcl = erfcl

#include "libm.h"
#include "longdouble.h"

static long double tiny = 1e-40L,
        nearunfl = 1e-4000L,
        half = 0.5L,
        one = 1.0L,
        onehalf = 1.5L,
        L16_3 = 16.0L / 3.0L;

static long double
tiny        = 1e-40L,
nearunfl    = 1e-4000L,
half        = 0.5L,
one         = 1.0L,
onehalf     = 1.5L,
L16_3       = 16.0L/3.0L;
/*
 * Coefficients for even polynomial P for erf(x)=x+x*P(x^2) on [0,0.84375]
 */
static long double P[] = {              /* 21 coeffs */
        1.283791670955125738961589031215451715556e-0001L,
        -3.761263890318375246320529677071815594603e-0001L,
```

```
        1.128379167095512573896158903121205899135e-0001L,
        -2.686617064513125175943235483344625046092e-0002L,
        5.223977625442187842111846652980454568389e-0003L,
        -8.548327023450852832546626271083862724358e-0004L,
        1.205533298178966425102164715902231976672e-0004L,
        -1.492565035840625097674944905027897838996e-0005L,
        1.646211436588924733604648849172936692024e-0006L,
        -1.636584469123491976815834704799733514987e-0007L,
        1.480719281587897445302529007144770739305e-0008L,
        -1.229055530170782843046467986464722047175e-0009L,
        9.422759064320307357553954945760654341633e-0011L,
        -6.711366846653439036162105104991433380926e-0012L,
        4.463224090341893165100275380693843116240e-0013L,
        -2.783513452582658245422635662559779162312e-0014L,
        1.634227412586960195251346878863754661546e-0015L,
        -9.060782672889577722765711455623117802795e-0017L,
        4.741341801266246873412159213893613602354e-0018L,
        -2.272417596497826188374846636534317381203e-0019L,
        8.069088737316068462496835658928566920933e-0021L,
};
```

*unchanged_portion_omitted_*

```
#endif /* ! codereview */
static long double Q1[] = {     /*  12 bottom coeffs with leading 1.0 hidden */
        9.081506296064882195280178373107623196655e-0001L,
        6.821049531968204097604392183650687642520e-0001L,
        4.067869178233539502315055970743271822838e-0001L,
        1.702332233546316765818144723063881095577e-0001L,
        7.498098377690553934266423088708614219356e-0002L,
        2.050154396918178697056927234366372760310e-0002L,
        7.012988534031999899054782333851905939379e-0003L,
        1.149904787014400354649843451234570731076e-0003L,
        3.185620255011299476196039491205159718620e-0004L,
        1.273405072153008775426376193374105840517e-0005L,
        4.753866999959432971956781228148402971454e-0006L,
        -1.002287602111660026053981728549540200683e-0006L,
};

#endif /* ! codereview */
/*
 * Rational erf(x) = ((float)0.95478588343) + P2(x-1.5)/Q2(x-1.5)
 * on [1.25,1.75]
 */
static long double C2 = (long double)((float)0.95478588343);
static long double P2[] = {             /*  12 top coeffs */
        1.131926304864446730135126164594785863512e-0002L,
        1.273617996967754151544330055186210322832e-0001L,
        -8.169980734667512519897816907190281143423e-0002L,
        9.512267486090321197833634271787944271746e-0002L,
        -2.394251569804872160005274999735914368170e-0002L,
        1.108768660227528667525252333184520222905e-0002L,
        3.527435492933902414662043314373277494221e-0004L,
        4.946116273341953463584319006669474625971e-0004L,
        -4.289851942513144714600285769022420962418e-0005L,
        8.304719841341952705874781636002085119978e-0005L,
        -1.040460226177309338781902252282849903189e-0005L,
        2.122913331584921470381327583672044434087e-0006L,
};

#endif /* ! codereview */
static long double Q2[] = {     /*  13 bottom coeffs with leading 1.0 hidden */
        7.448815737306992749168727691042003832150e-0001L,
        7.161813850236008294484744312430122188043e-0001L,
        3.603134756584225766144922727405641236121e-0001L,
        1.955811609133766478080550795194535852653e-0001L,
        7.253059963716225972479693813787810711233e-0002L,
```



```
        2.752391253757421424212770221541238324978e-0002L,
        7.677654852085240257439050673446546828005e-0003L,
        2.141102244555509687346497060326630061069e-0003L,
        4.342123013830957093949563339130674364271e-0004L,
        8.664587895570043348530991997272212150316e-0005L,
        1.109201582511752087060167429397033701988e-0005L,
        1.357834375781831062713470000309843641 1e-0006L,
        4.957746280594384997273090385060680016451e-0008L,
};

#endif /* ! codereview */
/*
 * erfc(x) = exp(-x*x)/x * R1(1/x)/S1(1/x) on [1.75, 16/3]
 */
static long double R1[] = {             /*  14 top coeffs */
        4.630195122654315016370705767621550602948e+0006L,
        1.257949521746494830700654204488675713628e+0007L,
        1.704153822720260272814743497376181625707e+0007L,
        1.502600568706061872381577539537315739943e+0007L,
        9.543710793431995284827024445387333922861e+0006L,
        4.589344808584091011652238164935949522427e+0006L,
        1.714660662941745791190907071920671844289e+0006L,
        5.034802147768798894307672256192466283867e+0005L,
        1.162286400443554670553152110447126850725e+0005L,
        2.086643834548901681362757308058660399137e+0004L,
        2.839793161868140305907004392890348777338e+0003L,
        2.786687241658423601778258694498655680778e+0002L,
        1.779177837102695602425897452623985786464e+0001L,
        5.641895835477470769043614623819144434731e-0001L,
};

#endif /* ! codereview */
static long double S1[] = {     /* 15 bottom coeffs with leading 1.0 hidden */
        4.630195122654331529595606896287596843110e+0006L,
        1.780411093345512024324781084220509055058e+0007L,
        3.250113097051800703707108623715776848283e+0007L,
        3.737857099176755050912193712123489115755e+0007L,
        3.029787497516578821459174055870781168593e+0007L,
        1.833850619965384765005769632103205777227e+0007L,
        8.562719999736915722210391226391865864 98e+0006L,
        3.139684562074658971315545539760008136973e+0006L,
        9.106421313731384880027703627454366930945e+0005L,
        2.085108342384266508613267136003194920001e+0005L,
        3.723126272693120340730491416449539290600e+0004L,
        5.049169878567344046145695360784436929802e+0003L,
        4.944274532748010767670150730035392093899e+0002L,
        3.153510608818213929982940249162268971412e+0001L,
        1.0e00L,
};

/*
 * erfc(x) = exp(-x*x)/x * R2(1/x)/S2(1/x) on [16/3, 107]
 */
static long double R2[] = {     /*  15 top coeffs in reverse order!! */
static long double R2[] = {     /*  15 top coeffs in reverse order!!*/
        2.447288012254302966796326587537136931669e+0005L,
        8.768592567189861896653369912716538739016e+0005L,
        1.552293152581780065761497908005779524953e+0006L,
        1.792075924835942935864231657504259926729e+0006L,
        1.504001463155897344947500222052694835875e+0006L,
        9.699485556326891411801230186016013019935e+0005L,
        4.961449933661807969863435013364796037700e+0005L,
        2.048726544693474028061176764716228273791e+0005L,
        6.891532964330949722479061090551896886635e+0004L,
        1.888014709010307507771964047905823237985e+0004L,
        4.189692064988957745054734809642495644502e+0003L,
```

```
        7.36234648742704806821296888964274173462 1e+0002L,
        9.98035971421141142300764105658081311620 7e+0001L,
        9.42691089513537918110719196219348517415 9e+0000L,
        5.64189583547756286948079451562360128042 9e-0001L,
};

#endif /* ! codereview */
static long double S2[] = {             /* 16 coefficients */
        2.44728220360190297124600471679060468688 0e+0005L,
        1.15300985275938530936775946093480848983 3e+0006L,
        2.60858064961263913154896626507866338484 9e+0006L,
        3.76667391734662330885020279239056902574 0e+0006L,
        3.89056625513838391078992492054133537069 1e+0006L,
        3.05288207390074620761316625999415052773 2e+0006L,
        1.88557451997038098846024104724851941840 7e+0006L,
        9.36972203475994318585145084681144501292 2e+0005L,
        3.79227835053668611144486975262449244365 9e+0005L,
        1.25775060695011579996536600177309405872 0e+0005L,
        3.41083060024236937064560863464362035505 8e+0004L,
        7.51398446974234313485132686317506727124 0e+0003L,
        1.31329632059319000255477999813869550784 0e+0003L,
        1.77397270088762915700632633369689651676 9e+0002L,
        1.67087645182258680042200901388045709416 2e+0001L,
        1.000L,
};

long double
erfl(long double x)
long double erfl(x)
long double x;
{
        long double erfcl(long double), s, y, t;
        long double erfcl(long double),s,y,t;

        if (!finitel(x)) {
                if (x != x)
                        return (x + x);         /* NaN */

                return (copysignl(one, x));     /* return +-1.0 is x=Inf */
            if (x != x) return x+x;     /* NaN */
            return copysignl(one,x);    /* return +-1.0 is x=Inf */
        }

        y = fabsl(x);

#endif /* ! codereview */
        if (y <= 0.84375L) {
                if (y <= tiny)
                        return (x + P[0] * x);

                s = y * y;
                t = __poly_libmq(s, 21, P);
                return (x + x * t);
            if (y<=tiny) return x+P[0]*x;
            s = y*y;
            t = __poly_libmq(s,21,P);
            return  x+x*t;
        }

        if (y <= 1.25L) {
                s = y - one;
                t = C1 + __poly_libmq(s, 12, P1) / (one + s * __poly_libmq(s,
                    12, Q1));
                return ((signbitl(x)) ? -t : t);
        } else if (y <= 1.75L) {
                s = y - onehalf;
```



```
                t = C2 + __poly_libmq(s, 12, P2) / (one + s * __poly_libmq(s,
                    13, Q2));
                return ((signbitl(x)) ? -t : t);
        if (y<=1.25L) {
            s = y-one;
            t = C1+__poly_libmq(s,12,P1)/(one+s*__poly_libmq(s,12,Q1));
            return (signbitl(x))? -t: t;
        } else if (y<=1.75L) {
            s = y-onehalf;
            t = C2+__poly_libmq(s,12,P2)/(one+s*__poly_libmq(s,13,Q2));
            return (signbitl(x))? -t: t;
        }

        if (y <= 9.0L)
                t = erfcl(y);
        else
                t = tiny;

        return ((signbitl(x)) ? t - one : one - t);
        if (y<=9.0L) t = erfcl(y); else t = tiny;
        return (signbitl(x))? t-one: one-t;
}

long double
erfcl(long double x)
long double erfcl(x)
long double x;
{
        long double s, y, t;
        long double erfl(long double),s,y,t;

        if (!finitel(x)) {
                if (x != x)
                        return (x + x);         /* NaN */

                /*
                 * return 2.0 if x = -inf
                 *        0.0 if x = +inf
                 */
                if (x < 0.0L)
                        return (2.0L);
                else
                        return (0.0L);
            if (x != x) return x+x;     /* NaN */
                                        /* return 2.0 if x= -inf
                                                   0.0 if x= +inf */
            if (x<0.0L) return 2.0L; else return 0.0L;
        }

        if (x <= 0.84375L) {
                if (x <= 0.25)
                        return (one - erfl(x));

                s = x * x;
                t = half - x;
                t = t - x * __poly_libmq(s, 21, P);
                return (half + t);
            if (x<=0.25) return one-erfl(x);
            s = x*x;
            t = half-x;
            t = t - x*__poly_libmq(s,21,P);
            return  half+t;
        }

        if (x <= 1.25L) {
                s = x - one;
```

```
                t = one - C1;
                return (t - __poly_libmq(s, 12, P1) / (one + s * __poly_libmq(s,
                    12, Q1)));
        } else if (x <= 1.75L) {
                s = x - onehalf;
                t = one - C2;
                return (t - __poly_libmq(s, 12, P2) / (one + s * __poly_libmq(s,
                    13, Q2)));
        if (x<=1.25L) {
            s = x-one;
            t = one-C1;
            return t - __poly_libmq(s,12,P1)/(one+s*__poly_libmq(s,12,Q1));
        } else if (x<=1.75L) {
            s = x-onehalf;
            t = one-C2;
            return t - __poly_libmq(s,12,P2)/(one+s*__poly_libmq(s,13,Q2));
        }

        if (x >= 107.0L) {
                return (nearunfl * nearunfl);   /* underflow */
        } else if (x >= L16_3) {
                y = __poly_libmq(x, 15, R2);
                t = y / __poly_libmq(x, 16, S2);
        if (x>=107.0L) return nearunfl*nearunfl;                /* underflow */
        else if (x >= L16_3) {
            y = __poly_libmq(x,15,R2);
            t = y/__poly_libmq(x,16,S2);
        } else {
                y = __poly_libmq(x, 14, R1);
                t = y / __poly_libmq(x, 15, S1);
            y = __poly_libmq(x,14,R1);
            t = y/__poly_libmq(x,15,S1);
        }

#endif /* ! codereview */
        /* see comment in ../Q/erfl.c */
        y = x;
        *(int *)&y = 0;
        t *= expl(-y * y) * expl(-(x - y) * (x + y));
        return (t);
        *(int*)&y = 0;
        t *= expl(-y*y)*expl(-(x-y)*(x+y));
        return  t;
}
```

_____*unchanged_portion_omitted_*

```
**********************************************************
    1413 Thu Jun  6 20:38:15 2019
new/usr/src/lib/libm/common/LD/finitel.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#include "libm.h"

#if defined(__sparc)
int
finitel(long double x)
{
        int *px = (int *)&x;

finitel(long double x) {
        int *px = (int *) &x;
        return ((px[0] & ~0x80000000) < 0x7fff0000);
}
#elif defined(__x86)
int
finitel(long double x)
{
        int *px = (int *)&x, t = px[2] & 0x7fff;

finitel(long double x) {
        int *px = (int *) &x, t = px[2] & 0x7fff;
#if defined(HANDLE_UNSUPPORTED)
        return (t != 0x7fff && ((px[1] & 0x80000000) != 0 || t == 0));
#else
        return (t != 0x7fff);
#endif
}
_____unchanged_portion_omitted_
```

```
**********************************************************
    1321 Thu Jun  6 20:38:15 2019
new/usr/src/lib/libm/common/LD/gammal.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __gammal = gammal

/*
 * long double gammal(long double x);
 */

#include "libm.h"
#include "longdouble.h"

extern int signgam;
extern int signgaml;

long double
gammal(long double x)
{
gammal(long double x) {
        long double y = __k_lgammal(x, &signgaml);

        signgam = signgaml;     /* SUSv3 requires the setting of signgam */
        return (y);
        return y;
}
______unchanged_portion_omitted_
```

```
**********************************************************
    1234 Thu Jun  6 20:38:15 2019
new/usr/src/lib/libm/common/LD/gammal_r.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

/*
 * long double gammal_r(long double x, int *signgamlp);
 */

#pragma weak __gammal_r = gammal_r

#include "libm.h"
#include "longdouble.h"

long double
gammal_r(long double x, int *signgamlp)
{
        return (__k_lgammal(x, signgamlp));
gammal_r(long double x, int *signgamlp) {
        return __k_lgammal(x, signgamlp);
}
______unchanged_portion_omitted_
```

```
**********************************************************
    3754 Thu Jun  6 20:38:16 2019
new/usr/src/lib/libm/common/LD/hypotl.c
11210 libm should be cstyle(10NBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __hypotl = hypotl

/*
 * hypotl(x,y)
 * Method :
 *      If z=x*x+y*y has error less than sqrt(2)/2 ulp than sqrt(z) has
 *      error less than 1 ulp.
 *      So, compute sqrt(x*x+y*y) with some care as follows:
 *      Assume x>y>0;
 *      1. save and set rounding to round-to-nearest
 *      2. if x > 2y  use
 *              x1*x1+(y*y+(x2*(x+x2))) for x*x+y*y
 *      where x1 = x with lower 32 bits cleared, x2 = x-x1; else
 *      3. if x <= 2y use
 *              t1*y1+((x-y)*(x-y)+(t1*y2+t2*y))
 *      where t1 = 2x with lower 64 bits cleared, t2 = 2x-t1, y1= y with
 *      lower 32 bits cleared, y2 = y-y1.
 *
 *      NOTE: DO NOT remove parenthsis!
 *
 * Special cases:
 *      hypot(x,y) is INF if x or y is +INF or -INF; else
 *      hypot(x,y) is NAN if x or y is NAN.
 *
 * Accuracy:
 *      hypot(x,y) returns sqrt(x^2+y^2) with error less than 1 ulps (units
 *      in the last place)
 */

#include "libm.h"
```



```
#if defined(__x86)
extern enum fp_direction_type __swap87RD(enum fp_direction_type);

#define k       0x7fff

long double
hypotl(long double x, long double y)
{
hypotl(long double x, long double y) {
        long double t1, t2, y1, y2, w;
        int *px = (int *)&x, *py = (int *)&y;
        int *pt1 = (int *)&t1, *py1 = (int *)&y1;
        int *px = (int *) &x, *py = (int *) &y;
        int *pt1 = (int *) &t1, *py1 = (int *) &y1;
        enum fp_direction_type rd;
        int j, nx, ny, nz;

        px[2] &= 0x7fff;        /* clear sign bit and padding bits of x and y */
        py[2] &= 0x7fff;
        nx = px[2];             /* biased exponent of x and y */
        ny = py[2];

#endif /* ! codereview */
        if (ny > nx) {
                w = x;
                x = y;
                y = w;
                nz = ny;
                ny = nx;
                nx = nz;
        }                               /* force nx >= ny */

#endif /* ! codereview */
        if (nx - ny >= 66)
                return (x + y); /* x / y >= 2**65 */

#endif /* ! codereview */
        if (nx < 0x5ff3 && ny > 0x205b) {       /* medium x,y */
                /* save and set RD to Rounding to nearest */
                rd = __swap87RD(fp_nearest);
                w = x - y;

#endif /* ! codereview */
                if (w > y) {
                        pt1[2] = px[2];
                        pt1[1] = px[1];
                        pt1[0] = 0;
                        t2 = x - t1;
                        x = sqrtl(t1 * t1 - (y * (-y) - t2 * (x + t1)));
                } else {
                        x += x;
                        py1[2] = py[2];
                        py1[1] = py[1];
                        py1[0] = 0;
                        y2 = y - y1;
                        pt1[2] = px[2];
                        pt1[1] = px[1];
                        pt1[0] = 0;
                        t2 = x - t1;
                        x = sqrtl(t1 * y1 - (w * (-w) - (t2 * y1 + y2 * x)));
                }

#endif /* ! codereview */
                if (rd != fp_nearest)
                        __swap87RD(rd);         /* restore rounding mode */
```

```
#endif /* ! codereview */
                return (x);
        } else {
                if (nx == k || ny == k) {       /* x or y is INF or NaN */
                        /* since nx >= ny; nx is always k within this block */
                        if (px[1] == 0x80000000 && px[0] == 0)
                                return (x);
                        else if (ny == k && py[1] == 0x80000000 && py[0] == 0)
                                return (y);
                        else
                                return (x + y);
                }

#endif /* ! codereview */
                if (ny == 0) {
                        if (y == 0.L || x == 0.L)
                                return (x + y);

#endif /* ! codereview */
                        pt1[2] = 0x3fff + 16381;
                        pt1[1] = 0x80000000;
                        pt1[0] = 0;
                        py1[2] = 0x3fff - 16381;
                        py1[1] = 0x80000000;
                        py1[0] = 0;
                        x *= t1;
                        y *= t1;
                        return (y1 * hypotl(x, y));
                }

#endif /* ! codereview */
                j = nx - 0x3fff;
                px[2] -= j;
                py[2] -= j;
                pt1[2] = nx;
                pt1[1] = 0x80000000;
                pt1[0] = 0;
                return (t1 * hypotl(x, y));
        }
}
#endif
```

```
**********************************************************
    1592 Thu Jun  6 20:38:16 2019
new/usr/src/lib/libm/common/LD/isnanl.c
11210 libm should be cstyle(10NBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __isnanl = isnanl

#include "libm.h"

#if defined(__sparc)
int
isnanl(long double x)
{
        int *px = (int *)&x;

        return ((px[0] & ~0x80000000) >= 0x7fff0000 && ((px[0] & ~0xffff0000) |
            px[1] | px[2] | px[3]) != 0);
isnanl(long double x) {
        int *px = (int *) &x;
        return ((px[0] & ~0x80000000) >= 0x7fff0000 &&
                ((px[0] & ~0xffff0000) | px[1] | px[2] | px[3]) != 0);
}
#elif defined(__x86)
int
isnanl(long double x)
{
        int *px = (int *)&x, t = px[2] & 0x7fff;

isnanl(long double x) {
        int *px = (int *) &x, t = px[2] & 0x7fff;
#if defined(HANDLE_UNSUPPORTED)
        return ((t == 0x7fff && ((px[1] & ~0x80000000) | px[0]) != 0) || (t !=
            0 && (px[1] & 0x80000000) == 0));
        return ((t == 0x7fff && ((px[1] & ~0x80000000) | px[0]) != 0) ||
                (t != 0 && (px[1] & 0x80000000) == 0));
```



```
#else
        return (t == 0x7fff && ((px[1] & ~0x80000000) | px[0]) != 0);
#endif
}
_____unchanged_portion_omitted_
```

```
**********************************************************
   27601 Thu Jun  6 20:38:17 2019
new/usr/src/lib/libm/common/LD/j0l.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
   1 /*
   2  * CDDL HEADER START
   3  *
   4  * The contents of this file are subject to the terms of the
   5  * Common Development and Distribution License (the "License").
   6  * You may not use this file except in compliance with the License.
   7  *
   8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
   9  * or http://www.opensolaris.org/os/licensing.
  10  * See the License for the specific language governing permissions
  11  * and limitations under the License.
  12  *
  13  * When distributing Covered Code, include this CDDL HEADER in each
  14  * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  15  * If applicable, add the following below this CDDL HEADER, with the
  16  * fields enclosed by brackets "[]" replaced with your own identifying
  17  * information: Portions Copyright [yyyy] [name of copyright owner]
  18  *
  19  * CDDL HEADER END
  20  */

  22 /*
  23  * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
  24  */

  26 #endif /* ! codereview */
  27 /*
  28  * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
  29  * Use is subject to license terms.
  30  */

  32 /*
  33  * Floating point Bessel's function of the first and second kinds
  34  * of order zero: j0(x),y0(x);
  35  *
  36  * Special cases:
  37  *      y0(0)=y1(0)=yn(n,0) = -inf with division by zero signal;
  38  *      y0(-ve)=y1(-ve)=yn(n,-ve) are NaN with invalid signal.
  39  */

  41 #pragma weak __j0l = j0l
  42 #pragma weak __y0l = y0l

  44 #include "libm.h"

  46 #include "longdouble.h"

  48 #include <math.h>
  49 #if defined(__SUNPRO_C)
  50 #include <sunmath.h>
  51 #endif

  53 #define GENERIC long double

  55 static GENERIC zero = 0.0L,
  56         small = 1.0e-9L,
  57         tiny = 1.0e-38L,
  58         one = 1.0L,
  59         five = 5.0L,
  60         eight = 8.0L,
  61         invsqrtpi = 5.641895835477562869480794515607725858441e-0001L,
```



```
  62         tpi = 0.636619772367581343075535053490057448L;
  25 static GENERIC
  26 zero    = 0.0L,
  27 small   = 1.0e-9L,
  28 tiny    = 1.0e-38L,
  29 one     = 1.0L,
  30 five    = 5.0L,
  31 eight   = 8.0L,
  32 invsqrtpi = 5.641895835477562869480794515607725858441e-0001L,
  33 tpi     = 0.636619772367581343075535053490057448L;

  64 static GENERIC pzero(), qzero();

  66 #endif /* ! codereview */
  67 static GENERIC r0[7] = {
  68         -2.499999999999999999999999999999998934492e-0001L,
  69         1.272657927360049786327618451133763714880e-0002L,
  70         -2.694499763712963276900636693400659600898e-0004L,
  71         2.724877475058977576903234070919616447883e-0006L,
  72         -1.432617103214330236967477495393076320281e-0008L,
  73         3.823248804080079168706683540513792224471e-0011L,
  74         -4.183174277567983647337568504286313665065e-0014L,
  75 };

  77 #endif /* ! codereview */
  78 static GENERIC s0[7] = {
  79         1.0e0L,
  80         1.159368290559800854689526195462884666395e-0002L,
  81         6.629397597394973383009743876169946772559e-0005L,
  82         2.426779981394054406305431142501735094340e-0007L,
  83         6.097663491248511069094400469635449749883e-0010L,
  84         1.017019133340929220238747413216052224036e-0012L,
  85         9.012593179306197579518374581969371278481e-0016L,
  86 };

  88 GENERIC
  89 j0l(GENERIC x)
  90 {
  36 j0l(x) GENERIC x; {
  91         GENERIC z, s, c, ss, cc, r, u, v;
  92         int i;

  94         if (isnanl(x))
  95                 return (x + x);

  41                 return (x+x);
  97         x = fabsl(x);

  99 #endif /* ! codereview */
 100         if (x > 1.28L) {
 101                 if (!finitel(x))
 102                         return (zero);

 104 #endif /* ! codereview */
 105                 s = sinl(x);
 106                 c = cosl(x);

 108                 /* BEGIN CSTYLED */
 109 #endif /* ! codereview */
 110                 /*
 111                  * j0(x) = sqrt(2/(pi*x))*(p0(x)*cos(x0)-q0(x)*sin(x0))
 112                  * where x0 = x-pi/4
 113                  *      Better formula:
 114                  *              cos(x0) = cos(x)cos(pi/4)+sin(x)sin(pi/4)
 115                  *                      = 1/sqrt(2) * (cos(x) + sin(x))
 116                  *              sin(x0) = sin(x)cos(pi/4)-cos(x)sin(pi/4)
```

```
117                  *                       = 1/sqrt(2) * (sin(x) - cos(x))
118                  * To avoid cancellation, use
119                  *              sin(x) +- cos(x) = -cos(2x)/(sin(x) -+ cos(x))
120                  * to compute the worse one.
121                  */
122                 /* END CSTYLED */
123 #endif /* ! codereview */
124                 if (x > 1.0e2450L) {    /* x+x may overflow */
125                         ss = s - c;
126                         cc = s + c;
 43                         ss = s-c;
 44                         cc = s+c;
127                 } else if (signbitl(s) != signbitl(c)) {
128                         ss = s - c;
129                         cc = -cosl(x + x) / ss;
 47                         cc = -cosl(x+x)/ss;
130                 } else {
131                         cc = s + c;
132                         ss = -cosl(x + x) / cc;
 50                         ss = -cosl(x+x)/cc;
133                 }

135 #endif /* ! codereview */
136                 /*
137                  * j0(x) = 1/sqrt(pi) * (P(0,x)*cc - Q(0,x)*ss) / sqrt(x)
138                  * y0(x) = 1/sqrt(pi) * (P(0,x)*ss + Q(0,x)*cc) / sqrt(x)
139                  */
140                 if (x > 1.0e120L)
141                         return ((invsqrtpi * cc) / sqrtl(x));

143                 u = pzero(x);
144                 v = qzero(x);
145                 return (invsqrtpi * (u * cc - v * ss) / sqrtl(x));
 52                         return (invsqrtpi*cc)/sqrtl(x);
 53                 u = pzero(x); v = qzero(x);
 54                 return (invsqrtpi*(u*cc-v*ss)/sqrtl(x));
146         }

148 #endif /* ! codereview */
149         if (x <= small) {
150                 if (x <= tiny)
151                         return (one - x);
 56                         return (one-x);
152                 else
153                         return (one - x * x * 0.25L);
 58                         return (one-x*x*0.25L);
154         }

156         z = x * x;
157         r = r0[6];
158         s = s0[6];

 60         z = x*x;
 61         r = r0[6]; s = s0[6];
160         for (i = 5; i >= 0; i--) {
161                 r = r * z + r0[i];
162                 s = s * z + s0[i];
 63             r = r*z + r0[i];
 64             s = s*z + s0[i];
163         }

165         return (one + z * (r / s));
 66         return (one+z*(r/s));
166 }
_____unchanged_portion_omitted_
```



```
179 #endif /* ! codereview */
180 static GENERIC v0[8] = {
181         1.0e0L,
182         1.247164416539111311571676766127767127970e-0002L,
183         7.829144749639791500052900281489367443576e-0005L,
184         3.247126540422245330511218321013360336606e-0007L,
185         9.750516724789499678567062572549568447869e-0010L,
186         2.156713223173591212250543390258458098776e-0012L,
187         3.322169561597890004231482431236452752624e-0015L,
188         2.821213295314000924252226486305726805093e-0018L,
189 };

191 GENERIC
192 y0l(GENERIC x)
193 {
194         GENERIC z, s, c, ss, cc, u, v;
195         GENERIC d __unused;
196         int i;

198         if (isnanl(x))
199                 return (x + x);

 79                 return (x+x);
201         if (x <= zero) {
202                 if (x == zero)
203                         d = -one / (x - x);
 82                     d = -one/(x-x);
204                 else
205                         d = zero / (x - x);
 84                     d = zero/(x-x);
206         }

208 #endif /* ! codereview */
209         if (x > 1.28L) {
210                 if (!finitel(x))
211                         return (zero);

213 #endif /* ! codereview */
214                 s = sinl(x);
215                 c = cosl(x);

217                 /* BEGIN CSTYLED */
218 #endif /* ! codereview */
219                 /*
220                  * j0(x) = sqrt(2/(pi*x))*(p0(x)*cos(x0)-q0(x)*sin(x0))
221                  * where x0 = x-pi/4
222                  *      Better formula:
223                  *              cos(x0) = cos(x)cos(pi/4)+sin(x)sin(pi/4)
224                  *                      = 1/sqrt(2) * (cos(x) + sin(x))
225                  *              sin(x0) = sin(x)cos(pi/4)-cos(x)sin(pi/4)
226                  *                      = 1/sqrt(2) * (sin(x) - cos(x))
227                  * To avoid cancellation, use
228                  *              sin(x) +- cos(x) = -cos(2x)/(sin(x) -+ cos(x))
229                  * to compute the worse one.
230                  */
231                 /* END CSTYLED */
232 #endif /* ! codereview */
233                 if (x > 1.0e2450L) {    /* x+x may overflow */
234                         ss = s - c;
235                         cc = s + c;
 86                         ss = s-c;
 87                         cc = s+c;
236                 } else if (signbitl(s) != signbitl(c)) {
237                         ss = s - c;
238                         cc = -cosl(x + x) / ss;
 90                         cc = -cosl(x+x)/ss;
```

```
                } else {
                        cc = s + c;
                        ss = -cosl(x + x) / cc;
                        ss = -cosl(x+x)/cc;
                }

#endif /* ! codereview */
                /*
                 * j0(x) = 1/sqrt(pi*x) * (P(0,x)*cc - Q(0,x)*ss)
                 * y0(x) = 1/sqrt(pi*x) * (P(0,x)*ss + Q(0,x)*cc)
                 */
                if (x > 1.0e120L)
                        return ((invsqrtpi * ss) / sqrtl(x));
                        return (invsqrtpi*ss)/sqrtl(x);
                return (invsqrtpi*(pzero(x)*ss+qzero(x)*cc)/sqrtl(x));

                return (invsqrtpi * (pzero(x) * ss + qzero(x) * cc) / sqrtl(x));
#endif /* ! codereview */
        }

        if (x <= tiny)
                return (u0[0] + tpi * logl(x));

        z = x * x;
        u = u0[7];
        v = v0[7];

        if (x <= tiny) {
            return (u0[0] + tpi*logl(x));
        }
        z = x*x;
        u = u0[7]; v = v0[7];
        for (i = 6; i >= 0; i--) {
                u = u * z + u0[i];
                v = v * z + v0[i];
            u = u*z + u0[i];
            v = v*z + v0[i];
        }

        return (u / v + tpi * (j0l(x) * logl(x)));
        return (u/v + tpi*(j0l(x)*logl(x)));
}
```

*unchanged_portion_omitted_*

```
#endif /* ! codereview */
static GENERIC ps0[11] = {
        1.0e0L,
        1.06605192787727337642536582396755051268 7e+0003L,
        4.39173964716838159239917380432926635303 8e+0005L,
        9.07516226180134367180565829412388886788 4e+0007L,
        1.02318611851990475181958191207598599505 8e+0010L,
        6.42786186041422374634051537651273027506 1e+0011L,
        2.21086150323782358973548130362799340623 5e+0013L,
        3.94324733578429290591595684090181817798 9e+0014L,
        3.28372097677754514215020011064727000448 1e+0015L,
        1.04534691881275404890364564153872898675 9e+0016L,
        8.04345546806561890075059958429119368046 3e+0015L,
};

#endif /* ! codereview */
static GENERIC pr1[12] = {              /* [8 -- 16] */
        9.999999999999999999999784422701108683618e-0001L,
        6.796098532948334207755488692777907062894e+0002L,
        1.840036112605722168824530758797169836042e+0005L,
        2.598490483191916637264894340635847598122e+0007L,
        2.105774863242707025525730249472054578523e+0009L,
```



```
        1.015822044230542426666314997796944979959e+0011L,
        2.931557457008110436764077699944189071875e+0012L,
        4.962885121125457633655259224179322808824e+0013L,
        4.705424055148223269155430598563351566279e+0014L,
        2.294439854910747229152056080910427001110e+0015L,
        4.905531843137486691500950019322475458629e+0015L,
        3.187543169710339218793442542845735994565e+0015L,
};

#endif /* ! codereview */
static GENERIC ps1[14] = {
        1.0e0L,
        6.796801657948334207754571576066758180288e+0002L,
        1.840512891201300567325421059826676366447e+0005L,
        2.599777028312918975306252167127695075221e+0007L,
        2.107582572771047636846811284634244892537e+0009L,
        1.017275794694156108975782763889979940348e+0011L,
        2.938487645192463845428059755454762316011e+0012L,
        4.982512164735557054521042916182317924466e+0013L,
        4.737639900153703274792677468264564361437e+0014L,
        2.323398719123742743524249528275097100646e+0015L,
        5.033419107069210577868909797896984419391e+0015L,
        3.409036105931068609601317076759804716059e+0015L,
        7.505655364352679737585745147753521662166e+0013L,
        -9.976837153983688250780198248297109118313e+0012L,
};

#endif /* ! codereview */
static GENERIC pr2[12] = {              /* [5 -- 8 ] */
        9.999999999999999937857236789277366320220e-0001L,
        3.692848765268649571651602420376358849214e+0002L,
        5.373022067535476576926715900057760985410e+0004L,
        4.038738891191314969971504035057219430725e+0006L,
        1.728285706306940523397385566659762646999e+0008L,
        4.375400819645889911158688737206054788534e+0009L,
        6.598950418204912408375591217782088567076e+0010L,
        5.827182039183238492480275401520072793783e+0011L,
        2.884222642913492390887572414999490975844e+0012L,
        7.373278873797767721932837830628688632775e+0012L,
        8.338295457568973761205077964397969230489e+0012L,
        2.911383183467288345772308817209806922143e+0012L,
};

#endif /* ! codereview */
static GENERIC ps2[14] = {
        1.0e0L,
        3.693551890268649477288896267171993213102e+0002L,
        5.375607880998361502474715133828068514297e+0004L,
        4.042477764024108249744998625727863673 28e+0006L,
        1.731069838737016956685839588670132939513e+0008L,
        4.387147674049898778738226585935491417728e+0009L,
        6.628058659206537653495569405677158258165e+0010L,
        5.869659904164177740471685856367322160664e+0011L,
        2.919839445622817017058977559638969436383e+0012L,
        7.535314897696671402628203718612309253907e+0012L,
        8.696355561452933775733098597486106589 35e+0012L,
        3.216155103141537221173601557697083216257e+0012L,
        4.756857081068942248246880159213789086363e+0010L,
        -3.496356619666608032231074866481472824067e+0009L,
};

#endif /* ! codereview */
static GENERIC pr3[13] = {              /* [3.5 -- 5 ] */
        9.999999999999916693107285612398196588247e-0001L,
        2.263975921282917721194425320484974336945e+0002L,
        1.994358386744245848889492762781484199966e+0004L,
```

```
        8.980067458430542243559962493831661323168e+0005L,
        2.282213787521372663705567756420087553508e+0007L,
        3.409784374889063618250288699908375135923e+0008L,
        3.024380857401448589254343517589811711108e+0009L,
        1.571110368046740246895017121443082286379e+0010L,
        4.603187020243604632153685300463160593768e+0010L,
        7.087196453409712719449549280664058793403e+0010L,
        5.046196021776346356803687409644239065041e+0010L,
        1.287758439080165765709154276618854799932e+0010L,
        5.900679773415023433787846658096813590784e+0008L,
};

#endif /* ! codereview */
static GENERIC ps3[13] = {
        1.0e0L,
        2.264679046282855061328604619231774747116e+0002L,
        1.995939523988944553755653255389812103448e+0004L,
        8.993853144706348727038389967490183236820e+0005L,
        2.288326099634588843906989983704795468773e+0007L,
        3.424967100255240885169240956804790118282e+0008L,
        3.046311797972463991368023759640028910016e+0009L,
        1.589614961932826812790222479700797224003e+0010L,
        4.692406624527744816497089139325073939927e+0010L,
        7.320486495902008912866462849073108323948e+0010L,
        5.345945972828978289935309597742981360994e+0010L,
        1.444033091910423754121309915092247171008e+0010L,
        7.987714685115314668378957273824383610525e+0008L,
};

#endif /* ! codereview */
static GENERIC pr4[13] = {              /* [2.5, 3.5] */
        9.999999999867366779611187227477577122 60e-0001L,
        1.453824980703800559037873123568378845663e+0002L,
        8.097327216430682288267610447006508661032e+0003L,
        2.273847252038264370231169686380192662135e+0005L,
        3.561056728046211111354759998976985449622e+0006L,
        3.244933588800096378434627029369680378599e+0007L,
        1.740112392860717950376210038908476792588e+0008L,
        5.426170187455893285197878563881579269524e+0008L,
        9.490107486454362321004377336020526281371e+0008L,
        8.688872439428470049801714121070005313806e+0008L,
        3.673315853166437222811910656900123215515e+0008L,
        5.577770470359303305164877446339693270239e+0007L,
        1.540438642031689641308197880181291865714e+0006L,
};

#endif /* ! codereview */
static GENERIC ps4[13] = {              /* [2.5, 3.5] */
        1.0e0L,
        1.454528105698159439773035951959131799816e+0002L,
        8.107442215200392397172179900434987859618e+0003L,
        2.279390393778242887574177096606328994140e+0005L,
        3.576251625592252008424781111770934135844e+0006L,
        3.267909499056932631405942058670933813863e+0007L,
        1.760021515330805537499778238099704648805e+0008L,
        5.525553787667353981242060222587465726729e+0008L,
        9.769870295912820457889384082671269328511e+0008L,
        9.110582071004774279226905629624018008454e+0008L,
        3.981857678621955599371967680343918454345e+0008L,
        6.482404686230769399073192961667697036706e+0007L,
        2.210046943095878402443535460329391782298e+0006L,
};

#endif /* ! codereview */
static GENERIC pr5[13] = {              /* [1.777..., 2.5] */
        9.999999991149861079518178711446558806 99e-0001L,
```



```
        9.252583736048588342568344570315435947614e+0001L,
        3.218726757856078715214631502407386264637e+0003L,
        5.554009964621111656479588505862577040831e+0004L,
        5.269993115643664338253196944523510290175e+0005L,
        2.874613773778430691192912190618220544575e+0006L,
        9.133538151103658353874146919613442436035e+0006L,
        1.673067041410338922825193013077354249193e+0007L,
        1.706913873848398011744790289200151840498e+0007L,
        9.067766583853288534551600235576747618679e+0006L,
        2.216746733457884568532695355036338655872e+0006L,
        1.945753880802872541235703812722344514405e+0005L,
        3.132374412921948071539195638885330951749e+0003L,
};

#endif /* ! codereview */
static GENERIC ps5[13] = {              /* [1.777..., 2.5] */
        1.0e0L,
        9.259614983862181118883831670990340052982e+0001L,
        3.225125275462903384842124075132609290304e+0003L,
        5.575705362829101545292760055941855246492e+0004L,
        5.306049863037087855496170121958448492522e+0005L,
        2.907060758873509564309729903109018597215e+0006L,
        9.298059206584995898298257827131208539289e+0006L,
        1.720391071006963176836108026556547062980e+0007L,
        1.782614812922865190479394509487941920612e+0007L,
        9.708016389605273153536452032839879950155e+0006L,
        2.476495084688170096480215640962175140027e+0006L,
        2.363200660365585759668077790194604917187e+0005L,
        4.803239569848196077121203575704356936731e+0003L,
};

#endif /* ! codereview */
static GENERIC pr6[13] = {              /* [1.28, 1.777...] */
        9.999999969777095495998606925524322559556e-0001L,
        5.825486719466194430503283824096872219216e+0001L,
        1.248155491637757281915184824965379905380e+0003L,
        1.302093199842358609321338417071710477615e+0004L,
        7.353835804186292782840961999810543016039e+0004L,
        2.356471661113686180549195092555751341757e+0005L,
        4.350553267429009581632987060942780847101e+0005L,
        4.588762661876600638719159826652389418235e+0005L,
        2.675796398548523436544221045225290128611e+0005L,
        8.077649557108971388298292919988449940464e+0004L,
        1.117640459221306873519068741664054573776e+0004L,
        5.544400072396814695175787511557757885585e+0002L,
        5.072550541191480498431289089905822910718e+0000L,
};

#endif /* ! codereview */
static GENERIC ps6[13] = {              /* [1.28, 1.777...] */
        1.0e0L,
        5.832517925357165050639075848183613063291e+0001L,
        1.252144364743592128171256104364976466898e+0003L,
        1.310300234342216813579118022415585740772e+0004L,
        7.434667697093812197817292154032863632923e+0004L,
        2.398706595587719165726469002404004614711e+0005L,
        4.472737517625103157004869372427480602511e+0005L,
        4.786313523337761975294171429067037723611e+0005L,
        2.851161872872731228472536061865365370192e+0005L,
        8.891648269899148412331918021801385815586e+0004L,
        1.297097489535351517572978123584751042287e+0004L,
        7.096761640545975756202184143400469812618e+0002L,
        8.378049338590233325977702401733340820351e+0000L,
};

#endif /* ! codereview */
```

```
506 static GENERIC sixteen = 16.0L;
507 static GENERIC huge = 1.0e30L;

509 static GENERIC
510 pzero(x)
511     GENERIC x;
124 static GENERIC pzero(x)
125 GENERIC x;
512 {
513         GENERIC s, r, t, z;
514         int i;

516 #endif /* ! codereview */
517         if (x > huge)
518                 return (one);

520         t = one / x;
521         z = t * t;

129         t = one/x; z = t*t;
523         if (x > sixteen) {
524                 r = z * pr0[11] + pr0[10];
525                 s = ps0[10];

131             r = z*pr0[11]+pr0[10]; s = ps0[10];
527                 for (i = 9; i >= 0; i--) {
528                         r = z * r + pr0[i];
529                         s = z * s + ps0[i];
133                 r = z*r + pr0[i];
134                 s = z*s + ps0[i];
530                 }
531         } else if (x > eight) {
532                 r = pr1[11];
533                 s = ps1[11] + z * (ps1[12] + z * ps1[13]);

137             r = pr1[11]; s = ps1[11]+z*(ps1[12]+z*ps1[13]);
535                 for (i = 10; i >= 0; i--) {
536                         r = z * r + pr1[i];
537                         s = z * s + ps1[i];
139                 r = z*r + pr1[i];
140                 s = z*s + ps1[i];
538                 }
539         } else if (x > five) {          /* x > 5.0 */
540                 r = pr2[11];
541                 s = ps2[11] + z * (ps2[12] + z * ps2[13]);

143             r = pr2[11]; s = ps2[11]+z*(ps2[12]+z*ps2[13]);
543                 for (i = 10; i >= 0; i--) {
544                         r = z * r + pr2[i];
545                         s = z * s + ps2[i];
145                 r = z*r + pr2[i];
146                 s = z*s + ps2[i];
546                 }
547         } else if (x > 3.5L) {
548                 r = pr3[12];
549                 s = ps3[12];

149             r = pr3[12]; s = ps3[12];
551                 for (i = 11; i >= 0; i--) {
552                         r = z * r + pr3[i];
553                         s = z * s + ps3[i];
151                 r = z*r + pr3[i];
152                 s = z*s + ps3[i];
554                 }
555         } else if (x > 2.5L) {
556                 r = pr4[12];
```



```
557                 s = ps4[12];

155             r = pr4[12]; s = ps4[12];
559                 for (i = 11; i >= 0; i--) {
560                         r = z * r + pr4[i];
561                         s = z * s + ps4[i];
157                 r = z*r + pr4[i];
158                 s = z*s + ps4[i];
562                 }
563         } else if (x > (1.0L / 0.5625L)) {
564                 r = pr5[12];
565                 s = ps5[12];

160         } else if (x > (1.0L/0.5625L)) {
161             r = pr5[12]; s = ps5[12];
567                 for (i = 11; i >= 0; i--) {
568                         r = z * r + pr5[i];
569                         s = z * s + ps5[i];
163                 r = z*r + pr5[i];
164                 s = z*s + ps5[i];
570                 }
571         } else {                        /* assume x > 1.28 */
572                 r = pr6[12];
573                 s = ps6[12];

167             r = pr6[12]; s = ps6[12];
575                 for (i = 11; i >= 0; i--) {
576                         r = z * r + pr6[i];
577                         s = z * s + ps6[i];
169                 r = z*r + pr6[i];
170                 s = z*s + ps6[i];
578                 }
579         }

581         return (r / s);
173         return (r/s);
582 }


583 static GENERIC qr0[12] = {             /* [16, inf] */
584         -1.2499999999999999999999999999999999972972e-0001L,
585         -1.4251795955456705774143957625039915968 97e+0002L,
586         -6.3124996456259708455344602579362224072 19e+0004L,
587         -1.4113743264572083843151212436988144468 48e+0007L,
588         -1.7350342127588735814109847578607872528 42e+0009L,
589         -1.1997776475127894894218263424850552806 80e+0011L,
590         -4.5960253340816557144998604096991003736 44e+0012L,
591         -9.2625256282012841077929244770316533991 87e+0013L,
592         -8.8583947286850392453443988421806628676 39e+0014L,
593         -3.2675279536875348876237406227095059721 13e+0015L,
594         -2.6642229711863119675871293470294500620 19e+0015L,
595         3.4424640607239878695851800953445041002 04e+0014L,
596 };

598 #endif /* ! codereview */
599 static GENERIC qs0[11] = {
600         1.0e0L,
601         1.1407296139365364619315166100031856878 81e+0003L,
602         5.0566655104422993510091981864900858035 80e+0005L,
603         1.1320417638256427879439416505227181991 15e+0008L,
604         1.3945701118725816063926206782142464797 67e+0010L,
605         9.6779452181522647895344310795637443784 21e+0011L,
606         3.7311403278515368282251430588963485020 96e+0013L,
607         7.6127859510648692917228466810208816764 10e+0014L,
608         7.4760770164067648917301910048118639759 40e+0015L,
609         2.9512464826135920354215034271003938317 09e+0016L,
```

```
        3.108361803691811711136854587074302034901e+0016L,
};

#endif /* ! codereview */
static GENERIC qr1[12] = {              /* [8, 16 ] */
        -1.249999999999999999997949010383433818157e-0001L,
        -9.051215166393822640636752244895124126934e+0001L,
        -2.620782703428148837671179031904208303947e+0004L,
        -3.975571261553504457766177974508785790884e+0006L,
        -3.479029330759311306270072218074074994090e+0008L,
        -1.823955008124268573036216746186239829089e+0010L,
        -5.765932697111801375765156029221568664435e+0011L,
        -1.079843680798742592954002192417934779114e+0013L,
        -1.146893630504592739082205764611581332897e+0014L,
        -6.367016059683898464936104447282880704182e+0014L,
        -1.583109041961213490464459111903484209098e+0015L,
        -1.230149555764242473103128650135795639412e+0015L,
};

#endif /* ! codereview */
static GENERIC qs1[14] = {
        1.0e0L,
        7.246831508115058112438579847778014458432e+0002L,
        2.100854184439168518399383786306927037611e+0005L,
        3.192636418837951507430188285940994235122e+0007L,
        2.801558443383354674538443461124434216152e+0009L,
        1.475026997664373739293483927250653467487e+0011L,
        4.694486824913954608552363821799927145318e+0012L,
        8.890350100919200250838438709601547334021e+0013L,
        9.626844429082905144874701068760469752067e+0014L,
        5.541110744600460773528263862687521642140e+0015L,
        1.486500494789452556727470329232123096563e+0016L,
        1.415840104845959400365430773732093899210e+0016L,
        1.780866095241517418081312567239682336483e+0015L,
        -2.359230917384889357887631544079990129494e+0014L,
};

#endif /* ! codereview */
static GENERIC qr2[12] = {              /* [5, 8] */
        -1.249999999999999531937744362527772181614e-0001L,
        -4.944373897356969774839375977239241573966e+0001L,
        -7.728449175433465285314261650078450473909e+0003L,
        -6.262574329612752346336901434651220705903e+0005L,
        -2.900948220220943306027235217424380672732e+0007L,
        -7.988719647634192770463917157562874119535e+0008L,
        -1.318228171927181389547760026626357012375e+0010L,
        -1.282439773983029245309263271945424928196e+0011L,
        -7.050925570827818040186149940257918845138e+0011L,
        -2.021751882573871990004205616874202684429e+0012L,
        -2.592939962400668552384333900573812635658e+0012L,
        -1.038267109518891262840601514932972850326e+0012L,
};

#endif /* ! codereview */
static GENERIC qs2[14] = {
        1.0e0L,
        3.961358492885570003202784022894248952116e+0002L,
        6.205788738864701882828752634586510926968e+0004L,
        5.045715603932670286550673813011764406749e+0006L,
        2.349248611362658323353343389430968751429e+0008L,
        6.520244524415828635917683553721880063911e+0009L,
        1.089111211223507719337067159886281887722e+0011L,
        1.080406000905359867958779409414903018610e+0012L,
        6.135645280895514703514154680623769562148e+0012L,
        1.862433040246625874245867151368643668215e+0013L,
        2.667780805786648888840777888702193708994e+0013L,
```



```
        1.394401107289087774765300711809313112824e+0013L,
        1.093247500616320375562898297156722445484e+0012L,
        -7.228875530378928722826604216491493780775e+0010L,
};

#endif /* ! codereview */
static GENERIC qr3[13] = {              /* [3.5 5] */
        -1.249999999999473067748420379578481661075e-0001L,
        -3.044549048635289351913574324803250977998e+0001L,
        -2.890081140649769078496693003524681440869e+0003L,
        -1.404922456817202235879343275330529107684e+0005L,
        -3.862746614385573443518177403617349281869e+0006L,
        -6.257517309110249049201133708911155047689e+0007L,
        -6.031451330920839916987079782727323477520e+0008L,
        -3.411542405173830611454025765755854382346e+0009L,
        -1.089392478149726672133014498723021526099e+0010L,
        -1.824934078420210941290140903415956782726e+0010L,
        -1.400780278304358710423481070486939531139e+0010L,
        -3.716484136064917363926635716743771092093e+0009L,
        -1.397591075296425529970434890954904331580e+0008L,
};

#endif /* ! codereview */
static GENERIC qs3[13] = {
        1.0e0L,
        2.441498613904962049391000187014945858042e+0002L,
        2.326188882072370711500164222341514337043e+0004L,
        1.137138213121231338494977104659239578165e+0006L,
        3.152918070735662728722998452605364253517e+0007L,
        5.172877993426507259314270488444013595108e+0008L,
        5.083086439731669807455961078856470774115e+0009L,
        2.961842732066434123119325521139476909941e+0010L,
        9.912185866862440735829781856081353151390e+0010L,
        1.793560561251622234430564181567297983598e+0011L,
        1.577090119341228122522652651084979404030 73e+0011L,
        5.509910306780166194333889999985463681636e+0010L,
        4.761691134078874491202320181517936758141e+0009L,
};

#endif /* ! codereview */
static GENERIC qr4[13] = {              /* [2.5 3.5] */
        -1.249999999928567734339745043490705340835e-0001L,
        -1.967201748731419063051601624435565528481e+0001L,
        -1.186329146714562236407099740615528170707e+0003L,
        -3.607736959222941810356301491152457934060e+0004L,
        -6.119200717978104904932828468575194267125e+0005L,
        -6.037847781158358226670305078652205586384e+0006L,
        -3.503558153336140359700536720393565984740e+0007L,
        -1.180196478268225718757218523746787309773e+0008L,
        -2.221860232085134915841426363505169680528e+0008L,
        -2.173372505452747585296176761701746236760e+0008L,
        -9.649364865061237558517730539506568013963e+0007L,
        -1.465429227847933034546039640094862650385e+0007L,
        -3.083003197920262085170581866246663380607e+0005L,
};

#endif /* ! codereview */
static GENERIC qs4[13] = {              /* [2.5 3.5] */
        1.0e0L,
        1.579620773732259142752614142139986854055e+0002L,
        9.581372220329138733203879503753685054968e+0003L,
        2.939598672379108095776114131010825885308e+0005L,
        5.052183049314542218630341818692588448168e+0006L,
        5.083497695595206639433839326338971980149e+0007L,
        3.036385361800553388049719014005099206516e+0008L,
        1.067826481452753409910563785161661492137e+0009L,
```

```
        2.1456441255571180447207417751253196692 72e+0009L,
        2.3241156159597199493639466734915522167 99e+0009L,
        1.2232629621120707579669598556198470111 46e+0009L,
        2.5697655533184954237384785859471102707 09e+0008L,
        1.3547447442992271278979057877326365655 04e+0007L,
};

#endif /* ! codereview */
static GENERIC qr5[13] = {              /* [1.777.., 2.5] */
        -1.2499999959366396976376804281745760699 71e-0001L,
        -1.2608460553713114534858919234264890683 15e+0001L,
        -4.7723984675444674808011743302901415788 95e+0002L,
        -8.9398525999902984866137608339964905997 24e+0003L,
        -9.1840707871495420509795422264461342431 97e+0004L,
        -5.4060389450182744583626378977392804351 71e+0005L,
        -1.8458965447051902610186537286781710844 18e+0006L,
        -3.6136169906808095018786675706533080715 47e+0006L,
        -3.9087829781356932522525577204143486237 79e+0006L,
        -2.1737110225173239271091381705884427681 76e+0006L,
        -5.4312531306799184858364085490078562444 95e+0005L,
        -4.5910985464526845100825915872759407659 59e+0004L,
        -5.2447113641682078068355200577922296465 78e+0002L,
};

#endif /* ! codereview */
static GENERIC qs5[13] = {              /* [1.777.., 2.5] */
        1.0e0L,
        1.0145362108512908783508927509724748614 47e+0002L,
        3.8755475106871353140644341600961396810 76e+0003L,
        7.3619131226700798149552592819956177325 80e+0004L,
        7.7202889442187711265810865395855293146 36e+0005L,
        4.6815295544467524964044314336083065580 38e+0006L,
        1.6678826219405039254550312523083677458 20e+0007L,
        3.4694031537613998818882726208553051562 41e+0007L,
        4.0969920479642107118670893847199478630 19e+0007L,
        2.5968047558292174493115307359595606305 54e+0007L,
        7.9839337746978892381544650640194107638 45e+0006L,
        9.8181338169799008190872424252807579381 52e+0005L,
        3.0610839308686943960135415356707454435 60e+0004L,
};

static GENERIC qr6[13] = {              /* [1.28, 1.777..] */
        -1.2499998815772890018071372828249290827 71e-0001L,
        -7.9982735100531107596108105941195336192 82e+0000L,
        -1.8724819553351725433690896177715656327 19e+0002L,
        -2.1221167867263008050798740033037996468 12e+0003L,
        -1.2938502858395292825031782634847734784 57e+0004L,
        -4.4450247422663161810333541922625293560 93e+0004L,
        -8.7301613783343577676683444673565053470 70e+0004L,
        -9.7062228951720784428014449725053150547 36e+0004L,
        -5.8963255182598582701655315136181953210 41e+0004L,
        -1.8231720343681088222764208270746688323 3e+0004L,
        -2.5093041786350559266388330403374723871 75e+0003L,
        -1.1566089657157792373167698289417299640 99e+0002L,
        -7.0280057896507313968873468263977852104 42e-0001L,
};

#endif /* ! codereview */
static GENERIC qs6[13] = {              /* [1.28, 1.777..] */
        1.0e0L,
        6.4572110850580648456012613212777210759 00e+0001L,
        1.5340052165880112103428245551360086829 50e+0003L,
        1.7772179991764417825933576604623790971 71e+0004L,
        1.1183726526424694680910848102632311996 96e+0005L,
        4.0152424338584618131423657483864736052 94e+0005L,
        8.3770810455170986454486165143882804976 73e+0005L,
```



```
        1.0114950200080103525753980096041642873 37e+0006L,
        6.8867220752904305686522278752002089559 70e+0005L,
        2.5047351899480214720471571486131719565 37e+0005L,
        4.4081389201710448469410018443520098170 62e+0004L,
        3.1055721780721151456730587228536408548 84e+0003L,
        5.5882948211189161134373965045738170336 78e+0001L,
};

static GENERIC
qzero(x)
    GENERIC x;
static GENERIC qzero(x)
GENERIC x;
{
        GENERIC s, r, t, z;
        int i;

#endif /* ! codereview */
        if (x > huge)
                return (-0.125L / x);

        t = one / x;
        z = t * t;

                return (-0.125L/x);
        t = one/x; z = t*t;
        if (x > sixteen) {
                r = z * qr0[11] + qr0[10];
                s = qs0[10];

            r = z*qr0[11]+qr0[10]; s = qs0[10];
                for (i = 9; i >= 0; i--) {
                        r = z * r + qr0[i];
                        s = z * s + qs0[i];
                r = z*r + qr0[i];
                s = z*s + qs0[i];
                }
        } else if (x > eight) {
                r = qr1[11];
                s = qs1[11] + z * (qs1[12] + z * qs1[13]);

            r = qr1[11]; s = qs1[11]+z*(qs1[12]+z*qs1[13]);
                for (i = 10; i >= 0; i--) {
                        r = z * r + qr1[i];
                        s = z * s + qs1[i];
                r = z*r + qr1[i];
                s = z*s + qs1[i];
                }
        } else if (x > five) {          /* assume x > 5.0 */
                r = qr2[11];
                s = qs2[11] + z * (qs2[12] + z * qs2[13]);

            r = qr2[11]; s = qs2[11]+z*(qs2[12]+z*qs2[13]);
                for (i = 10; i >= 0; i--) {
                        r = z * r + qr2[i];
                        s = z * s + qs2[i];
                r = z*r + qr2[i];
                s = z*s + qs2[i];
                }
        } else if (x > 3.5L) {
                r = qr3[12];
                s = qs3[12];

            r = qr3[12]; s = qs3[12];
                for (i = 11; i >= 0; i--) {
                        r = z * r + qr3[i];
```

```
860                         s = z * s + qs3[i];
219                 r = z*r + qr3[i];
220                 s = z*s + qs3[i];
861                 }
862         } else if (x > 2.5L) {
863                 r = qr4[12];
864                 s = qs4[12];

223             r = qr4[12]; s = qs4[12];
866                 for (i = 11; i >= 0; i--) {
867                         r = z * r + qr4[i];
868                         s = z * s + qs4[i];
225                 r = z*r + qr4[i];
226                 s = z*s + qs4[i];
869                 }
870         } else if (x > (1.0L / 0.5625L)) {
871                 r = qr5[12];
872                 s = qs5[12];

228         } else if (x > (1.0L/0.5625L)) {
229             r = qr5[12]; s = qs5[12];
874                 for (i = 11; i >= 0; i--) {
875                         r = z * r + qr5[i];
876                         s = z * s + qs5[i];
231                 r = z*r + qr5[i];
232                 s = z*s + qs5[i];
877                 }
878         } else {                        /* assume x > 1.28 */
879                 r = qr6[12];
880                 s = qs6[12];

235             r = qr6[12]; s = qs6[12];
882                 for (i = 11; i >= 0; i--) {
883                         r = z * r + qr6[i];
884                         s = z * s + qs6[i];
237                 r = z*r + qr6[i];
238                 s = z*s + qs6[i];
885                 }
886         }

888         return (t * (r / s));
241         return (t*(r/s));
889 }
```

_____*unchanged_portion_omitted_*

```
**********************************************************
   27182 Thu Jun  6 20:38:17 2019
new/usr/src/lib/libm/common/LD/j1l.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

/*
 * floating point Bessel's function of the first and second kinds
 * of order zero: j1(x),y1(x);
 *
 * Special cases:
 *      y0(0)=y1(0)=yn(n,0) = -inf with division by zero signal;
 *      y0(-ve)=y1(-ve)=yn(n,-ve) are NaN with invalid signal.
 */

#pragma weak __j1l = j1l
#pragma weak __y1l = y1l

#include "libm.h"
#include "longdouble.h"
#include <math.h>
#if defined(__SUNPRO_C)
#include <sunmath.h>
#endif

#define GENERIC long double

static GENERIC zero = 0.0L,
        small = 1.0e-9L,
        tiny = 1.0e-38L,
        one = 1.0L,
        five = 5.0L,
        invsqrtpi = 5.641895835477562869480794515607725858441e-0001L,
        tpi = 0.636619772367581343075535053490057448L;
static GENERIC
zero    = 0.0L,
```



```
small   = 1.0e-9L,
tiny    = 1.0e-38L,
one     = 1.0L,
five    = 5.0L,
invsqrtpi = 5.641895835477562869480794515607725858441e-0001L,
tpi     = 0.636619772367581343075535053490057448L;

static GENERIC pone(), qone();

#endif /* ! codereview */
static GENERIC r0[7] = {
        -6.249999999999999999999999999999999627320e-0002L,
        1.940606727194041716205384618494641565464e-0003L,
        -3.005630423155733701856481469986459043883e-0005L,
        2.345586219403918667468341047369572169358e-0007L,
        -9.976809285885253587529010109133336669724e-0010L,
        2.218743258363623946078958783775107473381e-0012L,
        -2.071079656218700604767650924103578046280e-0015L,
};

#endif /* ! codereview */
static GENERIC s0[7] = {
        1.0e0L,
        1.061695903156199920738051277075003059555e-0002L,
        5.521860513111180371566951179398862692060e-0005L,
        1.824214367413754193524107877084979441407e-0007L,
        4.098957778439576834818838198039029353925e-0010L,
        6.047735079699666389853240090925264056197e-0013L,
        4.679044728878836197247923279512047035041e-0016L,
};

GENERIC
j1l(GENERIC x)
{
j1l(x) GENERIC x; {
        GENERIC z, d, s, c, ss, cc, r;
        int i, sgn;

        if (!finitel(x))
                return (one / x);

                return (one/x);
        sgn = signbitl(x);
        x = fabsl(x);

#endif /* ! codereview */
        if (x > 1.28L) {
                s = sinl(x);
                c = cosl(x);

                /* BEGIN CSTYLED */
#endif /* ! codereview */
                /*
                 * j1(x) = sqrt(2/(pi*x))*(p1(x)*cos(x0)-q1(x)*sin(x0))
                 * where x0 = x-3pi/4
                 *      Better formula:
                 *              cos(x0) = cos(x)cos(3pi/4)+sin(x)sin(3pi/4)
                 *                      =  1/sqrt(2) * (sin(x) - cos(x))
                 *              sin(x0) = sin(x)cos(3pi/4)-cos(x)sin(3pi/4)
                 *                      = -1/sqrt(2) * (cos(x) + sin(x))
                 * To avoid cancellation, use
                 *              sin(x) +- cos(x) = -cos(2x)/(sin(x) -+ cos(x))
                 * to compute the worse one.
                 */
                /* END CSTYLED */
#endif /* ! codereview */
```

```
118                 if (x > 1.0e2450L) {    /* x+x may overflow */
119                         ss = -s - c;
120                         cc = s - c;
 43                         ss = -s-c;
 44                         cc =  s-c;
121                 } else if (signbitl(s) != signbitl(c)) {
122                         cc = s - c;
123                         ss = cosl(x + x) / cc;
 47                         ss = cosl(x+x)/cc;
124                 } else {
125                         ss = -s - c;
126                         cc = cosl(x + x) / ss;
 49                         ss = -s-c;
 50                         cc = cosl(x+x)/ss;
127                 }

129 #endif /* ! codereview */
130                 /*
131                  * j1(x) = 1/sqrt(pi*x) * (P(1,x)*cc - Q(1,x)*ss)
132                  * y1(x) = 1/sqrt(pi*x) * (P(1,x)*ss + Q(1,x)*cc)
133                  */
134                 if (x > 1.0e120L)
135                         return ((invsqrtpi * cc) / sqrtl(x));

137                 d = invsqrtpi * (pone(x) * cc - qone(x) * ss) / sqrtl(x);

 52                                         return (invsqrtpi*cc)/sqrtl(x);
 53                 d =  invsqrtpi*(pone(x)*cc-qone(x)*ss)/sqrtl(x);
139                 if (sgn == 0)
140                         return (d);
141                 else
142                         return (-d);
143         }

145 #endif /* ! codereview */
146         if (x <= small) {
147                 if (x <= tiny)
148                         d = 0.5L * x;
149                 else
150                         d = x * (0.5L - x * x * 0.125L);

 59             if (x <= tiny) d = 0.5L*x;
 60             else d =  x*(0.5L-x*x*0.125L);
152                 if (sgn == 0)
153                         return (d);
154                 else
155                         return (-d);
156         }

158         z = x * x;
 66         z = x*x;
159         r = r0[6];
160         s = s0[6];

162 #endif /* ! codereview */
163         for (i = 5; i >= 0; i--) {
164                 r = r * z + r0[i];
165                 s = s * z + s0[i];
 69                 r = r*z + r0[i];
 70                 s = s*z + s0[i];
166         }

168         d = x * 0.5L + x * (z * (r / s));

 72         d = x*0.5L+x*(z*(r/s));
170         if (sgn == 0)
```



```
171                 return (d);
172         else
173                 return (-d);
174 }
```

_____*unchanged_portion_omitted_*

```
186 #endif /* ! codereview */
187 static GENERIC v0[8] = {
188         1.0e0L,
189         1.369394302535807332517110204820556695644e-0002L,
190         9.508438148097659501433367062605935379588e-0005L,
191         4.399007309420092056052714797296467565655e-0007L,
192         1.488083087443756398305819693177715000787e-0009L,
193         3.751609832625793536245746965768587624922e-0012L,
194         6.680926434086257291872903276124244131448e-0015L,
195         6.676602383908906988160099057991121446058e-0018L,
196 };

198 GENERIC
199 y1l(GENERIC x)
200 {
 88 y1l(x) GENERIC x; {
201         GENERIC z, s, c, ss, cc, u, v;
202         int i;

204         if (isnanl(x))
205                 return (x + x);

 93                 return (x+x);
207         if (x <= zero) {
208                 if (x == zero)
209                         return (-one / zero);
 96                     return (-one/zero);
210                 else
211                         return (zero / zero);
 98                     return (zero/zero);
212         }

214 #endif /* ! codereview */
215         if (x > 1.28L) {
216                 if (!finitel(x))
217                         return (zero);

219 #endif /* ! codereview */
220                 s = sinl(x);
221                 c = cosl(x);

223                 /* BEGIN CSTYLED */
224 #endif /* ! codereview */
225                 /*
226                  * j1(x) = sqrt(2/(pi*x))*(p1(x)*cos(x0)-q1(x)*sin(x0))
227                  * where x0 = x-3pi/4
228                  *      Better formula:
229                  *              cos(x0) = cos(x)cos(3pi/4)+sin(x)sin(3pi/4)
230                  *                      =  1/sqrt(2) * (sin(x) - cos(x))
231                  *              sin(x0) = sin(x)cos(3pi/4)-cos(x)sin(3pi/4)
232                  *                      = -1/sqrt(2) * (cos(x) + sin(x))
233                  * To avoid cancellation, use
234                  *              sin(x) +- cos(x) = -cos(2x)/(sin(x) -+ cos(x))
235                  * to compute the worse one.
236                  */
237                 /* END CSTYLED */
238 #endif /* ! codereview */
239                 if (x > 1.0e2450L) {    /* x+x may overflow */
240                         ss = -s - c;
241                         cc = s - c;
```

```
100                         ss = -s-c;
101                         cc =  s-c;
242                 } else if (signbitl(s) != signbitl(c)) {
243                         cc = s - c;
244                         ss = cosl(x + x) / cc;
104                         ss = cosl(x+x)/cc;
245                 } else {
246                         ss = -s - c;
247                         cc = cosl(x + x) / ss;
106                         ss = -s-c;
107                         cc = cosl(x+x)/ss;
248                 }

250 #endif /* ! codereview */
251                 /*
252                  * j1(x) = 1/sqrt(pi*x) * (P(1,x)*cc - Q(1,x)*ss)
253                  * y1(x) = 1/sqrt(pi*x) * (P(1,x)*ss + Q(1,x)*cc)
254                  */
255                 if (x > 1.0e91L)
256                         return ((invsqrtpi * ss) / sqrtl(x));

258                 return (invsqrtpi * (pone(x) * ss + qone(x) * cc) / sqrtl(x));
109                         return (invsqrtpi*ss)/sqrtl(x);
110         return (invsqrtpi*(pone(x)*ss+qone(x)*cc)/sqrtl(x));
259         }

261         if (x <= tiny)
262                 return (-tpi / x);

264         z = x * x;
265         u = u0[6];
266         v = v0[6] + z * v0[7];

112         if (x <= tiny) {
113             return (-tpi/x);
114         }
115         z = x*x;
116             u = u0[6]; v = v0[6]+z*v0[7];
268         for (i = 5; i >= 0; i--) {
269                 u = u * z + u0[i];
270                 v = v * z + v0[i];
118                 u = u*z + u0[i];
119                 v = v*z + v0[i];
271         }

273         return (x * (u / v) + tpi * (j1l(x) * logl(x) - one / x));
121         return (x*(u/v) + tpi*(j1l(x)*logl(x)-one/x));
274 }
_____unchanged_portion_omitted_

291 #endif /* ! codereview */
292 static GENERIC ps0[11] = {
293         1.0e0L,
294         1.06060068754589145560218084327675809510 7e+0003L,
295         4.343106093416975589147153906505338900961e+0005L,
296         8.910605869002176566582072242244353399059e+0007L,
297         9.959122058635087888690713917622056540190e+0009L,
298         6.188744967234948231792482949171041843894e+0011L,
299         2.098863976953783506401759873801990304907e+0013L,
300         3.672870357018063196746729751479938908450e+0014L,
301         2.975538419246824921049011529574385888420e+0015L,
302         9.063657659950432050186860292844798370 91e+0015L,
303         6.401953344314747916729366441508892711691e+0015L,
304 };

306 #endif /* ! codereview */
```



```
307 static GENERIC pr1[12] = {
308         1.000000000000000000000236675241306609 84e+0000L,
309         6.746154419979618754354803488126452971204e+0002L,
310         1.811210781083390154857018330296145970502e+0005L,
311         2.533098390379924268038005329095287842244e+0007L,
312         2.029683619805342145252338570875424600729e+0009L,
313         9.660859662192711465301069401598929980319e+0010L,
314         2.743396238644831519934098967716621316316e+0012L,
315         4.553097354140854377931023170263455246288e+0013L,
316         4.210245069852219757476169864974870720374e+0014L,
317         1.987334056229596485076645967176169801727e+0015L,
318         4.067120052787096893838970455751338930462e+0015L,
319         2.486539606380406398310845264910691398133e+0015L,
320 };

322 #endif /* ! codereview */
323 static GENERIC ps1[14] = {
324         1.0e0L,
325         6.744982544979618754355808680196859521782e+0002L,
326         1.810421795396966762032155290441364740350e+0005L,
327         2.530986460644310651529583759699988435573e+0007L,
328         2.026743276048023121360249288818290224145e+0009L,
329         9.637461924407405935245269407052641341836e+0010L,
330         2.732378628423766417402292797028314160831e+0012L,
331         4.522345274960527124354844364012184278488e+0013L,
332         4.160650668341743132685335758415469856545e+0014L,
333         1.943730242988858208243492424892435901211e+0015L,
334         3.880228532692127989901131618598067450001e+0015L,
335         2.178020816161154615841000173683302999728e+0015L,
336         -8.994062666842225551554346698171600634173e+0013L,
337         1.368520368508851253495764806934619574990e+0013L,
338 };

340 #endif /* ! codereview */
341 static GENERIC pr2[12] = {
342         1.000000000000000006938651621840396237282e+0000L,
343         6.658416291850404981407101077037948144698e+0002L,
344         5.267073772170356547709794670602812447537e+0004L,
345         3.912012101226837463014925210735894620442e+0006L,
346         1.651295648974103957193874928714180765625e+0008L,
347         4.114901144480797609972484998142146783499e+0009L,
348         6.092524309766036681542980572526335147672e+0010L,
349         5.263913178071282616719249969074134570577e+0011L,
350         2.538408581124324223367341020538081330994e+0012L,
351         6.288607929360291027895126983015365677648e+0012L,
352         6.848330048211148419047055075386525945280e+0012L,
353         2.290309646838867941423178163991423244690e+0012L,
354 };

356 #endif /* ! codereview */
357 static GENERIC ps2[14] = {
358         1.0e0L,
359         3.657244416850405086459410165762319861856e+0002L,
360         5.262802358425023243992387075861237306312e+0004L,
361         3.905896813959919648136295861661483848364e+0006L,
362         1.646791907791461220742694842108202772763e+0008L,
363         4.096132803064256022224954120208201437344e+0009L,
364         6.046665195915950447544429445730680236759e+0010L,
365         5.198061739781991313414052212328653295168e+0011L,
366         2.484233851814333966401527626421254279796e+0012L,
367         6.047868806925315879339651539434315255940e+0012L,
368         6.333103831254091652501642567294101813354e+0012L,
369         1.875143098754284994467609936924685024968e+0012L,
370         -5.238330920563392692965412762508813601534e+0010L,
371         4.656888609439364725427789198383779259957e+0009L,
372 };
```

```
#endif /* ! codereview */
static GENERIC pr3[13] = {
        1.0000000000000093368873180680561378428 97e+0000L,
        2.2427199427284595884880515720028357291 83e+0002L,
        1.9554506113820265502662577373310956910 92e+0004L,
        8.7071432939936198993954005624091755907 39e+0005L,
        2.1862678944870045659483242890109545053 16e+0007L,
        3.2243285105419577923606915856675028646 88e+0008L,
        2.8210573551513805973317928968827413648 97e+0009L,
        1.4453713872954224043655847937960289798 40e+0010L,
        4.1817431606698913577830110026566581078 64e+0010L,
        6.3873710887679931193255361377945355139 22e+0010L,
        4.5756199994127160780640705877674164363 96e+0010L,
        1.2284156512116391606202844416905035508 42e+0010L,
        7.2421703498755630534360505321531128820 72e+0008L,
};

#endif /* ! codereview */
static GENERIC ps3[13] = {
        1.0e0L,
        2.2415480677285295510498046104860614010 70e+0002L,
        1.9528382167955521451321379329312371813 07e+0004L,
        8.6845749264931857446281273410699745755 26e+0005L,
        2.1763577710670379629408534128198521891 64e+0007L,
        3.1999586823561329773192587831671221005 67e+0008L,
        2.7862189315253346878446752199142018725 70e+0009L,
        1.4162837769517415496314175723179160397 67e+0010L,
        4.0429626592715679487356768346093488429 22e+0010L,
        6.0281684626466945100838472229684444021 61e+0010L,
        4.1184102267946414138338876065800852811 11e+0010L,
        9.9187357362970384307441612533382022302 63e+0009L,
        4.0929671982380980232191244874371303320 38e+0008L,
};

#endif /* ! codereview */
static GENERIC pr4[13] = {
        1.0000000000015092209781573990420595533 90e+0000L,
        1.4375518683781478511334999963237826077 87e+0002L,
        7.9113355374181772960415180614045054280 04e+0003L,
        2.1937109391153172147165189089357561048 04e+0005L,
        3.3906624951367309625134897965382749843 82e+0006L,
        3.0486553479293488910060706092938842747 89e+0007L,
        1.6137816334894966063540451615274509751 95e+0008L,
        4.9750898350372302771101561500384821599 88e+0008L,
        8.6360470870151154038809044183395663232 64e+0008L,
        7.9182029123283661401106712230769491015 09e+0008L,
        3.4232946657989847334396503117227948532 94e+0008L,
        5.6219049534419639610405039347826626136 21e+0007L,
        2.0863035433102402607586704045094844997 93e+0006L,
};

#endif /* ! codereview */
static GENERIC ps4[13] = {
        1.0e0L,
        1.4363799933845323716704933195918473623 04e+0002L,
        7.8946471547854306780610538488474366594 99e+0003L,
        2.1846597533920975290089817415508785861 74e+0005L,
        3.3661090833054651768035137381470494993 61e+0006L,
        3.0119115459689968176976658665872263431 86e+0007L,
        1.5822629137796898513167601484594148953 01e+0008L,
        4.8192688094943791921793858953013820177 0e+0008L,
        8.2013557629904506797028371234325271548 30e+0008L,
        7.2682320398251093741744642128234142521 2e+0008L,
        2.9509119090155729332621313239340364804 62e+0008L,
        4.2428399243059344230108589665406212193 96e+0007L,
```



```
        1.0643876204450907791821176663304051868 66e+0006L,
};

#endif /* ! codereview */
static GENERIC pr5[13] = {
        1.0000000001024348052411714272538473538 61e+0000L,
        9.1293322570836292590605022490259636234 21e+0001L,
        3.1322384835869530375761193775045571914 13e+0003L,
        5.3297825282693079712789431224541711078 61e+0004L,
        4.9884601571841177906928730021030529441 45e+0005L,
        2.6866020716157868161470103342560474693 78e+0006L,
        8.4454185260289611977037998087012683018 31e+0006L,
        1.5365753586461411574757258899079008273 90e+0007L,
        1.5684058182365238217968627705865448119 45e+0007L,
        8.4508762398887701023876186673623021735 47e+0006L,
        2.1544149001395673284240268271632034460 77e+0006L,
        2.1056569265650438988884602548080623522 05e+0005L,
        4.7391650110233965070221343037368628129 75e+0003L,
};

#endif /* ! codereview */
static GENERIC ps5[13] = {
        1.0e0L,
        9.1176135095953274765091526733947038477 93e+0001L,
        3.1216979724840156393012792292817707951 47e+0003L,
        5.2944472227358935680409118738345764402 55e+0004L,
        4.9303688821927723357982566841108878828 07e+0005L,
        2.6348546856411652983021674357983574377 68e+0006L,
        8.1854627754003263935558961570318182809 18e+0006L,
        1.4624174230802151926096686426630306670 86e+0007L,
        1.4506249939858516759828608441539548960 15e+0007L,
        7.4604676475619952832190865671620061138 64e+0006L,
        1.7542109814056124788692271425790563389 65e+0006L,
        1.4632867211552719715262649145247466995 96e+0005L,
        2.1558947257967020153412111165798270394 59e+0003L,
};

#endif /* ! codereview */
static GENERIC pr6[13] = {
        1.0000000035648555467417359203157431571 29e+0000L,
        5.7340039348625404581194235099095102883 66e+0001L,
        1.2095724919358504860865596922917968879 76e+0003L,
        1.2433983914222812479336747791636602868 38e+0004L,
        6.9309967551814379372582209986017082787 87e+0004L,
        2.1980676595327575986467222499667676200 99e+0005L,
        4.0336594327120586339331791158205768584 55e+0005L,
        4.2577596572190080270160472065745743586 78e+0005L,
        2.5119173958760043494807212774457639163 89e+0005L,
        7.8137561530706236541787316513818819535 52e+0004L,
        1.1520691733811278813855880929058643528 91e+0004L,
        6.5485807828040885537778160375515233980 82e+0002L,
        8.6687253701169061323275427661279384968 80e+0000L,
};

#endif /* ! codereview */
static GENERIC ps6[13] = {
        1.0e0L,
        5.7222852363571145664992215257362862051 84e+0001L,
        1.2030108428783179354445829506203395705 06e+0003L,
        1.2300583353785835501558255021724353712 08e+0004L,
        6.8009985506078612888653004386480898944 12e+0004L,
        2.1307678295993042629877693475368508859 21e+0005L,
        3.8404834666439166817599369729921553100 26e+0005L,
        3.9474323734592255428618191481080811603 93e+0005L,
        2.2378162393930811114815884344578385267 38e+0005L,
        6.5458204951244197233989462737909215407 74e+0004L,
```

```
        8.7295636303208927415007262132788347371 96e+0003L,
        4.1307626602918947534501747941969988137 09e+0002L,
        3.4803688986726846451303357860150755955 98e+0000L,
};

#endif /* ! codereview */
static GENERIC sixteen = 16.0L;
static GENERIC eight = 8.0L;
static GENERIC huge = 1.0e30L;

static GENERIC
pone(x)
    GENERIC x;
static GENERIC pone(x)
GENERIC x;
{
        GENERIC s, r, t, z;
        int i;

#endif /* ! codereview */
        if (x > huge)
                return (one);

        t = one / x;
        z = t * t;

        t = one/x; z = t*t;
        if (x > sixteen) {
                r = z * pr0[11] + pr0[10];
                s = ps0[10];

            r = z*pr0[11]+pr0[10]; s = ps0[10];
                for (i = 9; i >= 0; i--) {
                        r = z * r + pr0[i];
                        s = z * s + ps0[i];
                r = z*r + pr0[i];
                s = z*s + ps0[i];
                }
        } else if (x > eight) {
                r = pr1[11];
                s = ps1[11] + z * (ps1[12] + z * ps1[13]);

            r = pr1[11]; s = ps1[11]+z*(ps1[12]+z*ps1[13]);
                for (i = 10; i >= 0; i--) {
                        r = z * r + pr1[i];
                        s = z * s + ps1[i];
                r = z*r + pr1[i];
                s = z*s + ps1[i];
                }
        } else if (x > five) {
                r = pr2[11];
                s = ps2[11] + z * (ps2[12] + z * ps2[13]);

            r = pr2[11]; s = ps2[11]+z*(ps2[12]+z*ps2[13]);
                for (i = 10; i >= 0; i--) {
                        r = z * r + pr2[i];
                        s = z * s + ps2[i];
                r = z*r + pr2[i];
                s = z*s + ps2[i];
                }
        } else if (x > 3.5L) {
                r = pr3[12];
                s = ps3[12];

            r = pr3[12]; s = ps3[12];
                for (i = 11; i >= 0; i--) {
```



```
                        r = z * r + pr3[i];
                        s = z * s + ps3[i];
                r = z*r + pr3[i];
                s = z*s + ps3[i];
                }
        } else if (x > 2.5L) {
                r = pr4[12];
                s = ps4[12];

            r = pr4[12]; s = ps4[12];
                for (i = 11; i >= 0; i--) {
                        r = z * r + pr4[i];
                        s = z * s + ps4[i];
                r = z*r + pr4[i];
                s = z*s + ps4[i];
                }
        } else if (x > (1.0L / 0.5625L)) {
                r = pr5[12];
                s = ps5[12];

        } else if (x > (1.0L/0.5625L)) {
            r = pr5[12]; s = ps5[12];
                for (i = 11; i >= 0; i--) {
                        r = z * r + pr5[i];
                        s = z * s + ps5[i];
                r = z*r + pr5[i];
                s = z*s + ps5[i];
                }
        } else {                        /* assume x > 1.28 */
                r = pr6[12];
                s = ps6[12];

            r = pr6[12]; s = ps6[12];
                for (i = 11; i >= 0; i--) {
                        r = z * r + pr6[i];
                        s = z * s + ps6[i];
                r = z*r + pr6[i];
                s = z*s + ps6[i];
                }
        }

        return (r / s);
        return (r/s);
}


static GENERIC qr0[12] = {
        3.749999999999999999999999999999999971033e-0001L,
        4.256726035237050601607682277433094262226e+0002L,
        1.875976490812878489192409978945401066066e+0005L,
        4.170314268048041914273603680317745592790e+0007L,
        5.092750132543855817293451118974555746551e+0009L,
        3.494749676278488654103505795794139483404e+0011L,
        1.327062148257437316997667817096694173709e+0013L,
        2.648993138273427226907503742066551150490e+0014L,
        2.511695665909547412222430494473998127684e+0015L,
        9.274694506662289043224310499164702306096e+0015L,
        8.150904170663663829331320302911792892002e+0015L,
        -5.001918733707662355772037829620388765122e+0014L,
};

#endif /* ! codereview */
static GENERIC qs0[11] = {
        1.0e0L,
        1.135400380229880160428715273982155760093e+0003L,
        5.005701183877126164326765545516590744360e+0005L,
```

```
        1.1134442001137121679843376039330401029 87e+0008L,
        1.3610748199252230627787175656990394711 24e+0010L,
        9.3557509858028494844389339053259828096 53e+0011L,
        3.5634627860089888250039655438579980848 28e+0013L,
        7.1551451139000941636487268638038029104 54e+0014L,
        6.8712668358344727580555590138518436541 13e+0015L,
        2.6220308992267367126449749881573452340 92e+0016L,
        2.6029127291728763306500770217061397077 46e+0016L,
};

#endif /* ! codereview */
static GENERIC qr1[12] = {
        3.7499999999999999999977624582072844058 06e-0001L,
        2.6978839988817068399292555174981899804 85e+0002L,
        7.7551959257810284893869388704738344110 19e+0004L,
        1.1667777621040177771982110728955289683 55e+0007L,
        1.0115047729843211683200100845202610693 62e+0009L,
        5.2460077035741568535777545717202055500 10e+0010L,
        1.6376925498855926831661165516912665376 47e+0012L,
        3.0223036236981856699129903109250393824 95e+0013L,
        3.1547699272906556848461070302659099879 46e+0014L,
        1.7158199134415547700897309348081233609 21e+0015L,
        4.1650443557597326222735344451317361885 10e+0015L,
        3.1513814208741747056431003817080862875 96e+0015L,
};

#endif /* ! codereview */
static GENERIC qs1[14] = {
        1.0e0L,
        7.1970917053512182397856331724082769828 28e+0002L,
        2.0700127995995486855448830412976098610 55e+0005L,
        3.1170148153176562218718401527784587545 16e+0007L,
        2.7057196789025549748633258770259029717 27e+0009L,
        1.4061136147273457269250606487508672640 98e+0011L,
        4.4037775360671313203630059786316748173 59e+0012L,
        8.1707256902093222830614993867031672428 94e+0013L,
        8.6094588449754952892277941269644312105 66e+0014L,
        4.7667663670154734812572806006949529202 04e+0015L,
        1.2022865879433421948635579408881156416 50e+0016L,
        1.0124743283062009095250639360617560241 20e+0016L,
        6.1835520226789178582732228796158240707 03e+0014L,
       -9.7567315485582269975737374009882257227 40e+0013L,
};

#endif /* ! codereview */
static GENERIC qr2[12] = {
        3.7499999999999994812456472622269942931 89e-0001L,
        1.4713668072897713544911811401673590267 35e+0002L,
        2.2794324867684482201420809628435269512 50e+0004L,
        1.8289430485237712251638040433569582858 93e+0006L,
        8.3798283886478231358322205964177250108 37e+0007L,
        2.2798140293350440245853936712783780220 53e+0009L,
        3.7116539522571181208328177852714664414 20e+0010L,
        3.5576509145185545499167305725531050480 68e+0011L,
        1.9245834831460958962597743294989341606 50e+0012L,
        5.4243862560637363907595670882918871402 78e+0012L,
        6.8393256212417767862065097046717468417 37e+0012L,
        2.7021695631440011662916864523054363139 71e+0012L,
};

#endif /* ! codereview */
static GENERIC qs2[14] = {
        1.0e0L,
        3.9263791944393881357032119338952031910 89e+0002L,
        6.0891488041065982974883360630076093122 76e+0004L,
        4.8935461629732785837113763560416141506 45e+0006L,
```



```
        2.2475711191144978450463888018138322194 04e+0008L,
        6.1376356633501777512904693342007578726 45e+0009L,
        1.0051150197841028564244935195249989536 78e+0011L,
        9.7256644620145038328601513846046772406 20e+0011L,
        5.3455251004855111161486341928444346360 72e+0012L,
        1.5499440073989466917208627381739569947 79e+0013L,
        2.0671484411789526257103021241632647603 62e+0013L,
        9.4015654026419636112951194872425954623 01e+0012L,
        3.5482170886223982747488372877697093743 85e+0011L,
       -2.9344703417190471200765099384324173523 65e+0010L,
};

#endif /* ! codereview */
static GENERIC qr3[13] = {
        3.7499999999994127240845798332974514720 91e-0001L,
        9.0584785802917062124229784929384355825 27e+0001L,
        8.5240560331610387504610836667117243811 71e+0003L,
        4.1059671586291094277534345692236310147 30e+0005L,
        1.1183266033785313482597830919762333365 7e+0007L,
        1.7946366834035789185280649047141323293 43e+0008L,
        1.7143141574636359595561332360043688967 24e+0009L,
        9.6220920322360848465720672572676614560 30e+0009L,
        3.0577595244858591599577628587807683550 20e+0010L,
        5.1293067807547985316096214544159388900 20e+0010L,
        3.9991220027949610706806361943463160413 52e+0010L,
        1.1222984546434934859897215643581003453 88e+0010L,
        5.6039819876459897096688309685223625822 21e+0008L,
};

#endif /* ! codereview */
static GENERIC qs3[13] = {
        1.0e0L,
        2.4183286630765781698361551700536344199 22e+0002L,
        2.2796202059001210425875235412812728755 20e+0004L,
        1.1009842225857295214701290149922170927 94e+0006L,
        3.0107432236792470910042625162865545162 82e+0007L,
        4.8609255428273678172896192652155994339 96e+0008L,
        4.6866681110353486919827158643078395812 43e+0009L,
        2.6687017884051020174272147059467308940 74e+0010L,
        8.6773957461068026403905809448366058490 3e+0010L,
        1.5119364555749517906584987959451066430 36e+0011L,
        1.2608456044326234780020186968736083530 93e+0011L,
        4.0526922784198538539114402316008645898 05e+0010L,
        2.9655165192122260649832678222433296947 29e+0009L,
};

#endif /* ! codereview */
static GENERIC qr4[13] = {
        3.7499999999192341641546697544401230726 18e-0001L,
        5.8442185807768198647911682534850551018 58e+0001L,
        3.4892735140929129826756694113714356702 20e+0003L,
        1.0505236377745756845096634300189954795 94e+0005L,
        1.7645491720597015655007173197927801152 89e+0006L,
        1.7255324388441337950280631026814973711 54e+0007L,
        9.9381148473597785399651402475901763348 74e+0007L,
        3.3317108081845955453968837702007728423 14e+0008L,
        6.2719705576418815116095604448727972826 98e+0008L,
        6.1885297986773570750207749239037379132 85e+0008L,
        2.8219053027428499745099821678778850116 29e+0008L,
        4.6154673586469119767732902569843298148 96e+0007L,
        1.3481406087315464673966858026933806932 75e+0006L,
};

#endif /* ! codereview */
static GENERIC qs4[13] = {
        1.0e0L,
```

```
        1.561192663112345185261418296389902133372e+0002L,
        9.346678031144098270547225423124213083072e+0003L,
        2.825851246482293547838023847601704751590e+0005L,
        4.776572711622156091710902891124911556293e+0006L,
        4.715106953717135402977938048006267859302e+0007L,
        2.753962350894311316439652227611209035193e+0008L,
        9.428501434615463207768964787500411575223e+0008L,
        1.832650858775206787088236896454141572617e+0009L,
        1.901697378939743226948920874296595242257e+0009L,
        9.433322226854293780627188599226380812725e+0008L,
        1.808520540608671608680284520798858587370e+0008L,
        7.983342331736662753157217446919462398008e+0006L,
};

#endif /* ! codereview */
static GENERIC qr5[13] = {
        3.749999995331364437028988850515190446719e-0001L,
        3.739356381766559882677514593041627547911e+0001L,
        1.399562500629413529355265462912819802551e+0003L,
        2.594154053098947925345332218062210111753e+0004L,
        2.640149879297408640394163979394594318371e+0005L,
        1.542471854873199142031889093591449397995e+0006L,
        5.242272868972053374067572098992335425895e+0006L,
        1.025834487769410221329633071426044839935e+0007L,
        1.116553924239448940142230579060124209622e+0007L,
        6.318076065595910176374916303525884653514e+0006L,
        1.641218086168640408527639735915512881785e+0006L,
        1.522369793529178644168813882912134706444e+0005L,
        2.526530541062297200914180060208669584055e+0003L,
};

#endif /* ! codereview */
static GENERIC qs5[13] = {
        1.0e0L,
        9.998960735935075380397545659016287506660e+0001L,
        3.758767417842043742686475060540416737562e+0003L,
        7.013652806952306520121959742519780781653e+0004L,
        7.208949808818615099246529616211730446850e+0005L,
        4.272753927109614455417836186072202009252e+0006L,
        1.482524411356470699336129814111025434703e+0007L,
        2.988750366665678233425279237627700803473e+0007L,
        3.396957890261080492694709150553619185065e+0007L,
        2.050652487738593004111578091156304540386e+0007L,
        5.900504120811732547616511555946279451316e+0006L,
        6.563391409260160897024498082273183468347e+0005L,
        1.692629845012790205348966731477187041419e+0004L,
};

#endif /* ! codereview */
static GENERIC qr6[13] = {
        3.749999861516664133157566870858975421296e-0001L,
        2.367863756747764863120797431599473468918e+0001L,
        5.476715802114976248882067325630793143777e+0002L,
        6.143190357869842894025012945444096170251e+0003L,
        3.716250534677997850513733595140463851730e+0004L,
        1.270883463823876752138326905022875657430e+0005L,
        2.495301449636814481646371665429083801388e+0005L,
        2.789578988212952248340486296254398601942e+0005L,
        1.718247946911109055931819087137397324634e+0005L,
        5.458973214011665714330326732204106364229e+0004L,
        7.912102866879487860489433397595966652813e+0003L,
        4.077961006160866935722030715149087938091e+0002L,
        3.765206972770245085551057237882528510428e+0000L,
};

#endif /* ! codereview */
```



```
static GENERIC qs6[13] = {
        1.0e0L,
        6.341646532940517305641893852673926809601e+0001L,
        1.477058277414040790932597537920671025359e+0003L,
        1.674406564031044491436044253393536487604e+0004L,
        1.028516501369755949895050806908994650768e+0005L,
        3.593620042532885295087463507733285434207e+0005L,
        7.267924991381020915185873399453724799625e+0005L,
        8.462277510768818399961191426205006083088e+0005L,
        5.514399892230892163373611895645500250514e+0005L,
        1.898084241009259353540620272932188102299e+0005L,
        3.102941242117739015721984123081026253068e+0004L,
        1.958971184431466907681440650181421086143e+0003L,
        2.878853357310495087181721613889455121867e+0001L,
};

static GENERIC
qone(x)
    GENERIC x;
static GENERIC qone(x)
GENERIC x;
{
        GENERIC s, r, t, z;
        int i;

#endif /* ! codereview */
        if (x > huge)
                return (0.375L / x);

        t = one / x;
        z = t * t;

                return (0.375L/x);
        t = one/x; z = t*t;
        if (x > sixteen) {
                r = z * qr0[11] + qr0[10];
                s = qs0[10];

            r = z*qr0[11]+qr0[10]; s = qs0[10];
                for (i = 9; i >= 0; i--) {
                        r = z * r + qr0[i];
                        s = z * s + qs0[i];
                r = z*r + qr0[i];
                s = z*s + qs0[i];
                }
        } else if (x > eight) {
                r = qr1[11];
                s = qs1[11] + z * (qs1[12] + z * qs1[13]);

            r = qr1[11]; s = qs1[11]+z*(qs1[12]+z*qs1[13]);
                for (i = 10; i >= 0; i--) {
                        r = z * r + qr1[i];
                        s = z * s + qs1[i];
                r = z*r + qr1[i];
                s = z*s + qs1[i];
                }
        } else if (x > five) {          /* x > 5.0 */
                r = qr2[11];
                s = qs2[11] + z * (qs2[12] + z * qs2[13]);

            r = qr2[11]; s = qs2[11]+z*(qs2[12]+z*qs2[13]);
                for (i = 10; i >= 0; i--) {
                        r = z * r + qr2[i];
                        s = z * s + qs2[i];
                r = z*r + qr2[i];
                s = z*s + qs2[i];
```

```
 860                 }
 861         } else if (x > 3.5L) {
 862                 r = qr3[12];
 863                 s = qs3[12];

 231             r = qr3[12]; s = qs3[12];
 865                 for (i = 11; i >= 0; i--) {
 866                         r = z * r + qr3[i];
 867                         s = z * s + qs3[i];
 233                 r = z*r + qr3[i];
 234                 s = z*s + qs3[i];
 868                 }
 869         } else if (x > 2.5L) {
 870                 r = qr4[12];
 871                 s = qs4[12];

 237             r = qr4[12]; s = qs4[12];
 873                 for (i = 11; i >= 0; i--) {
 874                         r = z * r + qr4[i];
 875                         s = z * s + qs4[i];
 239                 r = z*r + qr4[i];
 240                 s = z*s + qs4[i];
 876                 }
 877         } else if (x > (1.0L / 0.5625L)) {
 878                 r = qr5[12];
 879                 s = qs5[12];

 242         } else if (x > (1.0L/0.5625L)) {
 243             r = qr5[12]; s = qs5[12];
 881                 for (i = 11; i >= 0; i--) {
 882                         r = z * r + qr5[i];
 883                         s = z * s + qs5[i];
 245                 r = z*r + qr5[i];
 246                 s = z*s + qs5[i];
 884                 }
 885         } else {                        /* assume x > 1.28 */
 886                 r = qr6[12];
 887                 s = qs6[12];

 249             r = qr6[12]; s = qs6[12];
 889                 for (i = 11; i >= 0; i--) {
 890                         r = z * r + qr6[i];
 891                         s = z * s + qs6[i];
 251                 r = z*r + qr6[i];
 252                 s = z*s + qs6[i];
 892                 }
 893         }

 895         return (t * (r / s));
 255         return (t*(r/s));
 896 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
    7147 Thu Jun  6 20:38:18 2019
new/usr/src/lib/libm/common/LD/jnl.c
11210 libm should be cstyle(10NBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __jnl = jnl
#pragma weak __ynl = ynl

/*
 * floating point Bessel's function of the 1st and 2nd kind
 * of order n: jn(n,x),yn(n,x);
 *
 * Special cases:
 *      y0(0)=y1(0)=yn(n,0) = -inf with division by zero signal;
 *      y0(-ve)=y1(-ve)=yn(n,-ve) are NaN with invalid signal.
 * Note 2. About jn(n,x), yn(n,x)
 *      For n=0, j0(x) is called,
 *      for n=1, j1(x) is called,
 *      for n<x, forward recursion us used starting
 *      from values of j0(x) and j1(x).
 *      for n>x, a continued fraction approximation to
 *      j(n,x)/j(n-1,x) is evaluated and then backward
 *      recursion is used starting from a supposed value
 *      for j(n,x). The resulting value of j(0,x) is
 *      compared with the actual value to correct the
 *      supposed value of j(n,x).
 *
 *      yn(n,x) is similar in all respects, except
 *      that forward recursion is used for all
 *      values of n>1.
 *
 */

#include "libm.h"
#include "longdouble.h"
```



```
#include <float.h>                      /* LDBL_MAX */

#define GENERIC long double

static const GENERIC
invsqrtpi = 5.641895835477562869480794515607725858441e-0001L,
        two = 2.0L,
        zero = 0.0L,
        one = 1.0L;
two  = 2.0L,
zero = 0.0L,
one  = 1.0L;

GENERIC
jnl(int n, GENERIC x)
{
        int i, sgn;
        GENERIC a, b, temp = 0, z, w;

        /*
         * J(-n,x) = (-1)^n * J(n, x), J(n, -x) = (-1)^n * J(n, x)
         * Thus, J(-n,x) = J(n,-x)
         */
        if (n < 0) {
                n = -n;
                x = -x;
        }

#endif /* ! codereview */
        if (n == 0)
                return (j0l(x));

#endif /* ! codereview */
        if (n == 1)
                return (j1l(x));

#endif /* ! codereview */
        if (x != x)
                return (x + x);

        if ((n & 1) == 0)
                return (x+x);
        if ((n&1) == 0)
                sgn = 0;                /* even n */
        else
                sgn = signbitl(x);      /* old n  */

#endif /* ! codereview */
        x = fabsl(x);

        if (x == zero || !finitel(x)) {
                b = zero;
        } else if ((GENERIC)n <= x) {
        if (x == zero || !finitel(x)) b = zero;
        else if ((GENERIC)n <= x) {
                /*
                 * Safe to use
                 * J(n+1,x)=2n/x *J(n,x)-J(n-1,x)
                 */
                if (x > 1.0e91L) {
                        /*
                         * x >> n**2
                         *  Jn(x) = cos(x-(2n+1)*pi/4)*sqrt(2/x*pi)
                         *  Yn(x) = sin(x-(2n+1)*pi/4)*sqrt(2/x*pi)
                         *  Let s=sin(x), c=cos(x),
                         *  xn=x-(2n+1)*pi/4, sqt2 = sqrt(2),then
```

```
 *
 *         n    sin(xn)*sqt2    cos(xn)*sqt2
 *      ----------------------------------
 *         0     s-c             c+s
 *         1    -s-c            -c+s
 *         2    -s+c            -c-s
 *         3     s+c             c-s
 */
switch (n & 3) {
switch (n&3) {
case 0:
        temp = cosl(x) + sinl(x);
        temp =  cosl(x)+sinl(x);
        break;
case 1:
        temp = -cosl(x) + sinl(x);
        temp = -cosl(x)+sinl(x);
        break;
case 2:
        temp = -cosl(x) - sinl(x);
        temp = -cosl(x)-sinl(x);
        break;
case 3:
        temp = cosl(x) - sinl(x);
        temp =  cosl(x)-sinl(x);
        break;
}

b = invsqrtpi * temp / sqrtl(x);
b = invsqrtpi*temp/sqrtl(x);
} else {
a = j0l(x);
b = j1l(x);

#endif /* ! codereview */
for (i = 1; i < n; i++) {
        temp = b;
        /* avoid underflow */
        b = b * ((GENERIC)(i + i) / x) - a;
        b = b*((GENERIC)(i+i)/x) - a;
        a = temp;
}
}
} else {
if (x < 1e-17L) {       /* use J(n,x) = 1/n!*(x/2)^n */
b = powl(0.5L * x, (GENERIC)n);

b = powl(0.5L*x, (GENERIC)n);
if (b != zero) {
        for (a = one, i = 1; i <= n; i++)
                a *= (GENERIC)i;

        b = b / a;
        b = b/a;
}
} else {
/* BEGIN CSTYLED */

#endif /* ! codereview */
/*
 * use backward recurrence
 *                        x      x^2      x^2
 *  J(n,x)/J(n-1,x) =  ----   ------   ------   .....
 *                        2n  - 2(n+1) - 2(n+2)
 *
 *                        1      1        1
```



```
 *  (for large x)   =  ----  ------   ------   .....
 *                      2n   2(n+1)   2(n+2)
 *                      -- - ------ - ------ -
 *                       x     x         x
 *
 * Let w = 2n/x and h=2/x, then the above quotient
 * is equal to the continued fraction:
 *                   1
 *      = -----------------------
 *                     1
 *         w - -----------------
 *                          1
 *              w+h - ---------
 *                     w+2h - ...
 *
 * To determine how many terms needed, let
 * Q(0) = w, Q(1) = w(w+h) - 1,
 * Q(k) = (w+k*h)*Q(k-1) - Q(k-2),
 * When Q(k) > 1e4      good for single
 * When Q(k) > 1e9      good for double
 * When Q(k) > 1e17     good for quaduple
 */

/*
 * END CSTYLED
 * determine k
 */
/* END CSTYLED */
/* determine k */
GENERIC t, v;
double q0, q1, h, tmp;
int k, m;

w = (n + n) / (double)x;
h = 2.0 / (double)x;
w  = (n+n)/(double)x;
h = 2.0/(double)x;
q0 = w;
z = w + h;
q1 = w * z - 1.0;
z = w+h;
q1 = w*z - 1.0;
k = 1;

#endif /* ! codereview */
while (q1 < 1.0e17) {
        k += 1;
        z += h;
        tmp = z * q1 - q0;
        tmp = z*q1 - q0;
        q0 = q1;
        q1 = tmp;
}

m = n + n;

for (t = zero, i = 2 * (n + k); i >= m; i -= 2)
        t = one / (i / x - t);

m = n+n;
for (t = zero, i = 2*(n+k); i >= m; i -= 2)
        t = one/(i/x-t);
a = t;
b = one;

#endif /* ! codereview */
```

```
234                         /*
235                          * Estimate log((2/x)^n*n!) = n*log(2/x)+n*ln(n)
236                          * hence, if n*(log(2n/x)) > ...
237                          *  single:
238                          *    8.8722839355e+01
239                          *  double:
240                          *    7.09782712893383973096e+02
241                          *  long double:
242                          *    1.13565234062941439494919310779707650 06170e+04
243                          * then recurrent value may overflow and the result is
244                          * likely underflow to zero.
245                          */
246                         tmp = n;
247                         v = two / x;
248                         tmp = tmp * logl(fabsl(v * tmp));

121                         v = two/x;
122                         tmp = tmp*logl(fabsl(v*tmp));
250                         if (tmp < 1.13565234062941439494919310779707 65e+04L) {
251                                 for (i = n - 1; i > 0; i--) {
124                                 for (i = n-1; i > 0; i--) {
252                                         temp = b;
253                                         b = ((i + i) / x) * b - a;
126                                         b = ((i+i)/x)*b - a;
254                                         a = temp;
255                                 }
256                         } else {
257                                 for (i = n - 1; i > 0; i--) {
130                                 for (i = n-1; i > 0; i--) {
258                                         temp = b;
259                                         b = ((i + i) / x) * b - a;
132                                         b = ((i+i)/x)*b - a;
260                                         a = temp;

262 #endif /* ! codereview */
263                                         if (b > 1e1000L) {
264                                                 a /= b;
265                                                 t /= b;
266                                                 b = 1.0;
267                                         }
268                                 }
269                         }

271                         b = (t * j0l(x) / b);
134                         b = (t*j0l(x)/b);
272                 }
273         }

275 #endif /* ! codereview */
276         if (sgn != 0)
277                 return (-b);
278         else
279                 return (b);
280 }

282 GENERIC
283 ynl(int n, GENERIC x)
284 {
285         int i;
286         int sign;
287         GENERIC a, b, temp = 0;

289         if (x != x)
290                 return (x + x);

137                 return (x+x);
```



```
292         if (x <= zero) {
293                 if (x == zero)
294                         return (-one / zero);
140                         return (-one/zero);
295                 else
296                         return (zero / zero);
142                         return (zero/zero);
297         }

299 #endif /* ! codereview */
300         sign = 1;

302 #endif /* ! codereview */
303         if (n < 0) {
304                 n = -n;

306                 if ((n & 1) == 1)
144                 if ((n&1) == 1)
307                         sign = -1;
308         }

310 #endif /* ! codereview */
311         if (n == 0)
312                 return (y0l(x));

314 #endif /* ! codereview */
315         if (n == 1)
316                 return (sign * y1l(x));

147                 return (sign*y1l(x));
318         if (!finitel(x))
319                 return (zero);

321         if (x > 1.0e91L) {
322                 /*
323                  * x >> n**2
324                  * Jn(x) = cos(x-(2n+1)*pi/4)*sqrt(2/x*pi)
325                  *   Yn(x) = sin(x-(2n+1)*pi/4)*sqrt(2/x*pi)
326                  *   Let s=sin(x), c=cos(x),
327                  * xn=x-(2n+1)*pi/4, sqt2 = sqrt(2),then
328                  *
329                  *         n    sin(xn)*sqt2    cos(xn)*sqt2
330                  *      ----------------------------------
331                  *         0     s-c             c+s
332                  *         1    -s-c            -c+s
333                  *         2    -s+c            -c-s
334                  *         3     s+c             c-s
335                  */
336                 switch (n & 3) {
166                 switch (n&3) {
337                 case 0:
338                         temp = sinl(x) - cosl(x);
168                         temp =  sinl(x)-cosl(x);
339                         break;
340                 case 1:
341                         temp = -sinl(x) - cosl(x);
171                         temp = -sinl(x)-cosl(x);
342                         break;
343                 case 2:
344                         temp = -sinl(x) + cosl(x);
174                         temp = -sinl(x)+cosl(x);
345                         break;
346                 case 3:
347                         temp = sinl(x) + cosl(x);
177                         temp =  sinl(x)+cosl(x);
348                         break;
```

```
349                 }

351                 b = invsqrtpi * temp / sqrtl(x);
180                 b = invsqrtpi*temp/sqrtl(x);
352         } else {
353                 a = y0l(x);
354                 b = y1l(x);

356 #endif /* ! codereview */
357                 /*
358                  * fix 1262058 and take care of non-default rounding
359                  */
360                 for (i = 1; i < n; i++) {
361                         temp = b;
362                         b *= (GENERIC)(i + i) / x;

184                         b *= (GENERIC) (i + i) / x;
364                         if (b <= -LDBL_MAX)
365                                 break;

367 #endif /* ! codereview */
368                         b -= a;
369                         a = temp;
370                 }
371         }

373 #endif /* ! codereview */
374         if (sign > 0)
375                 return (b);
376         else
377                 return (-b);
378 }
```

```
**********************************************************
    1325 Thu Jun  6 20:38:19 2019
new/usr/src/lib/libm/common/LD/lgammal.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __lgammal = lgammal

/*
 * long double lgammal(long double x);
 */

#include "libm.h"
#include "longdouble.h"

extern int signgam;
extern int signgaml;

long double
lgammal(long double x)
{
lgammal(long double x) {
        long double y = __k_lgammal(x, &signgaml);

        signgam = signgaml;     /* SUSv3 requires the setting of signgam */
        return (y);
        return y;
}
______unchanged_portion_omitted_
```

```
**********************************************************
    1238 Thu Jun  6 20:38:19 2019
new/usr/src/lib/libm/common/LD/lgammal_r.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

/*
 * long double lgammal_r(long double x, int *signgamlp);
 */

#pragma weak __lgammal_r = lgammal_r

#include "libm.h"
#include "longdouble.h"

long double
lgammal_r(long double x, int *signgamlp)
{
        return (__k_lgammal(x, signgamlp));
lgammal_r(long double x, int *signgamlp) {
        return __k_lgammal(x, signgamlp);
}
______unchanged_portion_omitted_
```

```
**********************************************************
    1599 Thu Jun  6 20:38:20 2019
new/usr/src/lib/libm/common/LD/log1pl.c
11210 libm should be cstyle(10NBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __log1pl = log1pl

/*
 * log1pl(x)
 * Kahan's trick based on log(1+x)/x being a slow varying function.
 */

#include "libm.h"

#if defined(__x86)
#define __swapRD        __swap87RD
#endif

#endif /* ! codereview */
extern enum fp_direction_type __swapRD(enum fp_direction_type);

long double
log1pl(long double x)
{
log1pl(long double x) {
        long double y;
        enum fp_direction_type rd;

        if (x != x)
                return (x + x);

#endif /* ! codereview */
        if (x < -1.L)
                return (logl(x));
```



```
#endif /* ! codereview */
        rd = __swapRD(fp_nearest);
        y = 1.L + x;

#endif /* ! codereview */
        if (y != 1.L) {
                if (y == x)
                        x = logl(x);
                else
                        x *= logl(y) / (y - 1.L);
        }

#endif /* ! codereview */
        if (rd != fp_nearest)
                (void) __swapRD(rd);

#endif /* ! codereview */
        return (x);
}
```

```
**********************************************************
    2492 Thu Jun  6 20:38:20 2019
new/usr/src/lib/libm/common/LD/logbl.c
11210 libm should be cstyle(10NBLD) clean
**********************************************************
   1 /*
   2  * CDDL HEADER START
   3  *
   4  * The contents of this file are subject to the terms of the
   5  * Common Development and Distribution License (the "License").
   6  * You may not use this file except in compliance with the License.
   7  *
   8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
   9  * or http://www.opensolaris.org/os/licensing.
  10  * See the License for the specific language governing permissions
  11  * and limitations under the License.
  12  *
  13  * When distributing Covered Code, include this CDDL HEADER in each
  14  * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  15  * If applicable, add the following below this CDDL HEADER, with the
  16  * fields enclosed by brackets "[]" replaced with your own identifying
  17  * information: Portions Copyright [yyyy] [name of copyright owner]
  18  *
  19  * CDDL HEADER END
  20  */

  22 /*
  23  * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
  24  */

  26 #endif /* ! codereview */
  27 /*
  28  * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
  29  * Use is subject to license terms.
  30  */

  32 #pragma weak __logbl = logbl

  34 #include "libm.h"
  35 #include "xpg6.h"                        /* __xpg6 */
  36 #define _C99SUSv3_logb          _C99SUSv3_logb_subnormal_is_like_ilogb

  38 #if defined(__sparc)
  39 #define ISNORMALL(k, x)         (k != 0x7fff)   /* assuming k != 0 */
  40 #define X86PDNRM(k, x)
  41 #define XSCALE_OFFSET           0x406f          /* 0x3fff + 112 */

  43 #endif /* ! codereview */
  44 static const long double xscale = 5192296858534827628530496329220096.0L;

  46 /* 2^112 */
  25                                                                  /* 2^112 */
  47 #elif defined(__x86)
  48 /*
  49  * if pseudo-denormal, replace by the equivalent normal
  50  */
  51 #define X86PDNRM(k, x)          if (k == 0 && (((int *)&x)[1] & 0x80000000) != \
  52             0) \
  53                 ((int *)&x)[2] |= k = 1
  30 #define X86PDNRM(k, x)  if (k == 0 && (((int *) &x)[1] & 0x80000000) != 0) \
  31                         ((int *) &x)[2] |= k = 1
  54 #if defined(HANDLE_UNSUPPORTED)         /* assuming k != 0 */
  55 #define ISNORMALL(k, x)         (k != 0x7fff && \
  56         (((int *)&x)[1] & 0x80000000) != 0)
  33 #define ISNORMALL(k, x) (k != 0x7fff && (((int *) &x)[1] & 0x80000000) != 0)
  57 #else
```



```
  58 #define ISNORMALL(k, x)         (k != 0x7fff)
  59 #endif
  60 #define XSCALE_OFFSET           0x403e                  /* 0x3fff + 63 */
  61 static const long double xscale = 9223372036854775808.0L;       /* 2^63 */
  62 #endif

  64 static long double
  65 raise_division(long double v)
  66 {
  42 raise_division(long double v) {
  67 #pragma STDC FENV_ACCESS ON

  69 #endif /* ! codereview */
  70         static const long double zero = 0.0L;

  72 #endif /* ! codereview */
  73         return (v / zero);
  74 }

  76 long double
  77 logbl(long double x)
  78 {
  44 logbl(long double x) {
  79         int k = XBIASED_EXP(x);

  81         X86PDNRM(k, x);

  83 #endif /* ! codereview */
  84         if (k == 0) {
  85                 if (ISZEROL(x)) {
  48                 if (ISZEROL(x))
  86                         return (raise_division(-1.0L));
  87                 } else if ((__xpg6 & _C99SUSv3_logb) != 0) {
  50                 else if ((__xpg6 & _C99SUSv3_logb) != 0) {
  88                         x *= xscale;    /* scale up by 2^112 or 2^63 */
  89                         return ((long double)(XBIASED_EXP(x) - XSCALE_OFFSET));
  90                 } else {
  91                         return ((long double)(-16382));
  92                 }
  93         } else if (ISNORMALL(k, x)) {
  94                 return ((long double)(k - 0x3fff));
  95         } else {
  52                         return (long double) (XBIASED_EXP(x) - XSCALE_OFFSET);
  53                 } else
  54                         return ((long double) (-16382));
  55         } else if (ISNORMALL(k, x))
  56                 return ((long double) (k - 0x3fff));
  57         else
  96                 return (x * x);
  97         }
  98 #endif /* ! codereview */
  99 }
```

```
**********************************************************
    6460 Thu Jun  6 20:38:20 2019
new/usr/src/lib/libm/common/LD/longdouble.h
11210 libm should be cstyle(10NBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#ifndef _LD_LONGDOUBLE_H
#define _LD_LONGDOUBLE_H
#include <sys/ieeefp.h>

extern long double __k_cosl(long double, long double);
extern long double __k_lgammal(long double, int *);
extern long double __k_sincosl(long double, long double, long double *);
extern long double __k_sinl(long double, long double);
extern long double __k_tanl(long double, long double, int);
extern long double __poly_libmq(long double, int, long double *);
extern int __rem_pio2l(long double, long double *);

extern long double acosdl(long double);
extern long double acoshl(long double);
extern long double acosl(long double);
extern long double acospil(long double);
extern long double acospl(long double);
extern long double aintl(long double);
extern long double anintl(long double);
extern long double annuityl(long double, long double);
extern long double asindl(long double);
extern long double asinhl(long double);
extern long double asinl(long double);
extern long double asinpil(long double);
extern long double asinpl(long double);
extern long double atan2dl(long double, long double);
extern long double atan2l(long double, long double);
extern long double atan2pil(long double, long double);
extern long double atandl(long double);
extern long double atanhl(long double);
```



```
extern long double atanl(long double);
extern long double atanpil(long double);
extern long double atanpl(long double);
extern long double cbrtl(long double);
extern long double ceill(long double);
extern long double compoundl(long double, long double);
extern long double copysignl(long double, long double);
extern long double cosdl(long double);
extern long double coshl(long double);
extern long double cosl(long double);
extern long double cospil(long double);
extern long double cospl(long double);
extern long double erfcl(long double);
extern long double erfl(long double);
extern long double exp10l(long double);
extern long double exp2l(long double);
extern long double expl(long double);
extern long double expm1l(long double);
extern long double fabsl(long double);
extern int finitel(long double);
extern long double floorl(long double);
extern long double fmodl(long double, long double);
extern enum fp_class_type fp_classl(long double);
extern long double gammal(long double);
extern long double hypotl(long double, long double);
extern int ilogbl(long double);
extern long double infinityl(void);
extern int irintl(long double);
extern int isinfl(long double);
extern int isnanl(long double);
extern int isnormall(long double);
extern int issubnormall(long double);
extern int iszerol(long double);
extern long double j0l(long double);
extern long double j1l(long double);
extern long double jnl(int, long double);
extern long double lgammal(long double);
extern long double log10l(long double);
extern long double log1pl(long double);
extern long double log2l(long double);
extern long double logbl(long double);
extern long double logl(long double);
extern long double max_normall(void);
extern long double max_subnormall(void);
extern long double min_normall(void);
extern long double min_subnormall(void);
extern long double nextafterl(long double, long double);
extern int nintl(long double);
extern long double pow_li(long double *, int *);
extern long double powl(long double, long double);
extern long double quiet_nanl(long);
extern long double remainderl(long double, long double);
extern long double rintl(long double);
extern long double scalbl(long double, long double);
extern long double scalbnl(long double, int);
extern long double signaling_nanl(long);
extern int signbitl(long double);
extern long double significandl(long double);
extern void sincosdl(long double, long double *, long double *);
extern void sincosl(long double, long double *, long double *);
extern void sincospil(long double, long double *, long double *);
extern void sincospl(long double, long double *, long double *);
extern long double sindl(long double);
extern long double sinhl(long double);
extern long double sinl(long double);
extern long double sinpil(long double);
```

```
extern long double sinpl(long double);
extern long double sqrtl(long double);
extern long double tandl(long double);
extern long double tanhl(long double);
extern long double tanl(long double);
extern long double tanpil(long double);
extern long double tanpl(long double);
extern long double y0l(long double);
extern long double y1l(long double);
extern long double ynl(int, long double);

extern long double q_copysign_(long double *, long double *);
extern long double q_fabs_(long double *);
extern int iq_finite_(long double *);
extern long double q_fmod_(long double *, long double *);
extern enum fp_class_type iq_fp_class_(long double *);
extern int iq_ilogb_(long double *);
extern long double q_infinity_(void);
extern int iq_isinf_(long double *);
extern int iq_isnan_(long double *);
extern int iq_isnormal_(long double *);
extern int iq_issubnormal_(long double *);
extern int iq_iszero_(long double *);
extern long double q_max_normal_(void);
extern long double q_max_subnormal_(void);
extern long double q_min_normal_(void);
extern long double q_min_subnormal_(void);
extern long double q_nextafter_(long double *, long double *);
extern long double q_quiet_nan_(long *);
extern long double q_remainder_(long double *, long double *);
extern long double q_scalbn_(long double *, int *);
extern long double q_signaling_nan_(long *);
extern int iq_signbit_(long double *);

#endif /* _LD_LONGDOUBLE_H */
```

```
**********************************************************
    2741 Thu Jun  6 20:38:21 2019
new/usr/src/lib/libm/common/LD/nextafterl.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __nextafterl = nextafterl

#include "libm.h"
#include <float.h>              /* LDBL_MAX, LDBL_MIN */

#if defined(__sparc)
#define n0      0
#define n1      1
#define n2      2
#define n3      3
#define X86PDNRM1(x)
#define INC(px) { \
                        if (++px[n3] == 0) \
                                if (++px[n2] == 0) \
                                        if (++px[n1] == 0) \
                                                ++px[n0]; \
                }
#define DEC(px) { \
                        if (--px[n3] == 0xffffffff) \
                                if (--px[n2] == 0xffffffff) \
                                        if (--px[n1] == 0xffffffff) \
                                                --px[n0]; \
                }
#elif defined(__x86)
#define         n0      2
#define         n1      1
#define         n2      0
#define         n3      0
/*
 * if pseudo-denormal, replace by the equivalent normal
 */
#define X86PDNRM1(x)    if (XBIASED_EXP(x) == 0 && (((int *)&x)[1] & \
```



```
#define X86PDNRM1(x)    if (XBIASED_EXP(x) == 0 && (((int *) &x)[1] & \
                                0x80000000) != 0) \
                                ((int *)&x)[2] |= 1
                                ((int *) &x)[2] |= 1
#define INC(px) { \
                        if (++px[n2] == 0) \
                                if ((++px[n1] & ~0x80000000) == 0) \
                                        px[n1] = 0x80000000, ++px[n0]; \
                }
#define DEC(px) { \
                        if (--px[n2] == 0xffffffff) \
                                if (--px[n1] == 0x7fffffff) \
                                        if ((--px[n0] & 0x7fff) != 0) \
                                                px[n1] |= 0x80000000; \
                }
#endif

long double
nextafterl(long double x, long double y)
{
        int *px = (int *)&x;
        int *py = (int *)&y;
nextafterl(long double x, long double y) {
        int *px = (int *) &x;
        int *py = (int *) &y;

        if (x == y)
                return (y);             /* C99 requirement */
        if (x != x || y != y)
                return (x * y);

        if (ISZEROL(x)) {       /* x == 0.0 */
                px[n0] = py[n0] & XSGNMSK;
                px[n1] = px[n2] = 0;
                px[n3] = 1;
        } else {
                X86PDNRM1(x);
                if ((px[n0] & XSGNMSK) == 0) {  /* x > 0.0 */
                        if (x > y)      /* x > y */
                                DEC(px)
                        else
                                INC(px)
                } else {
                        if (x < y)      /* x < y */
                                DEC(px)
                        else
                                INC(px)
                }
        }
#ifndef lint
        {
                volatile long double dummy __unused;
                int k = XBIASED_EXP(x);

                if (k == 0)
                        dummy = LDBL_MIN * copysignl(LDBL_MIN, x);
                else if (k == 0x7fff)
                        dummy = LDBL_MAX * copysignl(LDBL_MAX, x);
        }
#endif
        return (x);
}
______unchanged_portion_omitted_
```

```
**********************************************************
    1788 Thu Jun  6 20:38:21 2019
new/usr/src/lib/libm/common/LD/scalbl.c
11210 libm should be cstyle(10NBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __scalbl = scalbl

/*
 * scalbl(x,n): return x * 2**n by manipulating exponent.
 */

#include "libm.h"
#include "longdouble.h"

#include <sys/isa_defs.h>

long double
scalbl(long double x, long double fn)
{
        int *py = (int *)&fn, n;
scalbl(long double x, long double fn) {
        int *py = (int *) &fn, n;
        long double z;

        if (isnanl(x) || isnanl(fn))
                return (x * fn);
                return x * fn;

        /* fn is +/-Inf */
#if defined(_BIG_ENDIAN)
        if ((py[0] & 0x7fff0000) == 0x7fff0000) {
                if ((py[0] & 0x80000000) != 0)
#else
        if ((py[2] & 0x7fff) == 0x7fff) {
                if ((py[2] & 0x8000) != 0)
```



```
#endif
                        return (x / (-fn));
                        return x / (-fn);
                else
                        return (x * fn);
                        return x * fn;
        }

#endif /* ! codereview */
        if (rintl(fn) != fn)
                return ((fn - fn) / (fn - fn));

        if (fn > 65000.0L) {
                return (fn - fn) / (fn - fn);
        if (fn > 65000.0L)
                z = scalbnl(x, 65000);
        } else if (-fn > 65000.0L) {
        else if (-fn > 65000.0L)
                z = scalbnl(x, -65000);
        } else {
                n = (int)fn;
        else {
                n = (int) fn;
                z = scalbnl(x, n);
        }

        return (z);
        return z;
}
```

_____unchanged_portion_omitted_

```
**********************************************************
    1095 Thu Jun  6 20:38:22 2019
new/usr/src/lib/libm/common/LD/signgaml.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __signgaml = signgaml

#include "libm.h"
#include "longdouble.h"

int signgaml = 0;
```

```
**********************************************************
    1227 Thu Jun  6 20:38:22 2019
new/usr/src/lib/libm/common/LD/significandl.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __significandl = significandl

#include "libm.h"

long double
significandl(long double x)
{
significandl(long double x) {
        if (ISZEROL(x) || XBIASED_EXP(x) == 0x7fff)     /* 0/+-Inf/NaN */
                return (x + x);
        else
                return (scalbnl(x, -ilogbl(x)));
}
______unchanged_portion_omitted_
```

```
**********************************************************
    2914 Thu Jun  6 20:38:23 2019
new/usr/src/lib/libm/common/LD/sincosl.c
11210 libm should be cstyle(10NBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __sincosl = sincosl

/* BEGIN CSTYLED */
/*
 * cosl(x)
/* INDENT OFF */
/* cosl(x)
 * Table look-up algorithm by K.C. Ng, November, 1989.
 *
 * kernel function:
 *      __k_sincosl     ... sin and cos function on [-pi/4,pi/4]
 *      __rem_pio2l     ... argument reduction routine
 *
 * Method.
 *      Let S and C denote the sin and cos respectively on [-PI/4, +PI/4].
 *      1. Assume the argument x is reduced to y1+y2 = x-k*pi/2 in
 *         [-pi/2 , +pi/2], and let n = k mod 4.
 *      2. Let S=S(y1+y2), C=C(y1+y2). Depending on n, we have
 *
 *          n        sin(x)      cos(x)        tan(x)
 *     ----------------------------------------------------------
 *          0          S           C             S/C
 *          1          C          -S            -C/S
 *          2         -S          -C             S/C
 *          3         -C           S            -C/S
 *     ----------------------------------------------------------
 *
 * Special cases:
 *      Let trig be any of sin, cos, or tan.
 *      trig(+-INF)  is NaN, with signals;
```



```
 *      trig(NaN)    is that NaN;
 *
 * Accuracy:
 *      computer TRIG(x) returns trig(x) nearly rounded.
 */
/* END CSTYLED */
/* INDENT ON */

#include "libm.h"
#include "longdouble.h"

#include <sys/isa_defs.h>

void
sincosl(long double x, long double *s, long double *c)
{
sincosl(long double x, long double *s, long double *c) {
        long double y[2], z = 0.0L;
        int n, ix;

#endif /* ! codereview */
#if defined(__i386) || defined(__amd64)
        int *px = (int *)&x;
        int *px = (int *) &x;
#endif

        /* trig(Inf or NaN) is NaN */
        if (!finitel(x)) {
                *s = *c = x - x;
                return;
        }

        /* High word of x. */
#if defined(__i386) || defined(__amd64)
        XTOI(px, ix);
#else
        ix = *(int *)&x;
        ix = *(int *) &x;
#endif

        /* |x| ~< pi/4 */
        ix &= 0x7fffffff;

        if (ix <= 0x3ffe9220) {
        if (ix <= 0x3ffe9220)
                *s = __k_sincosl(x, z, c);
        }

        /* argument reduction needed */
        else {
                n = __rem_pio2l(x, y);

#endif /* ! codereview */
                switch (n & 3) {
                case 0:
                        *s = __k_sincosl(y[0], y[1], c);
                        break;
                case 1:
                        *c = -__k_sincosl(y[0], y[1], s);
                        break;
                case 2:
                        *s = -__k_sincosl(y[0], y[1], c);
                        *c = -*c;
                        break;
                case 3:
                        *c = __k_sincosl(y[0], y[1], s);
```

```
                        *s = -*s;
                }
        }
}
```

```
**********************************************************
    6020 Thu Jun  6 20:38:23 2019
new/usr/src/lib/libm/common/LD/sincospil.c
11210 libm should be cstyle(10NBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

/*
 * void sincospil(long double x, long double *s, long double *c)
 * *s = sinl(pi*x); *c = cosl(pi*x);
 *
 * Algorithm, 10/17/2002, K.C. Ng
 * ------------------------------
 * Let y = |4x|, z = floor(y), and n = (int)(z mod 8.0) (displayed in binary).
 *      1. If y == z, then x is a multiple of pi/4. Return the following values:
 *              ----------------------------------------------------
 *                n  x mod 2    sin(x*pi)    cos(x*pi)   tan(x*pi)
 *              ----------------------------------------------------
 *               000  0.00       +0 ___       +1 ___       +0
 *               001  0.25       +\/0.5       +\/0.5       +1
 *               010  0.50       +1 ___       +0 ___       +inf
 *               011  0.75       +\/0.5       -\/0.5       -1
 *               100  1.00       -0 ___       -1 ___       +0
 *               101  1.25       -\/0.5       -\/0.5       +1
 *               110  1.50       -1 ___       -0 ___       +inf
 *               111  1.75       -\/0.5       +\/0.5       -1
 *              ----------------------------------------------------
 *      2. Otherwise,
 *              ----------------------------------------------------
 *                n     t        sin(x*pi)    cos(x*pi)   tan(x*pi)
 *              ----------------------------------------------------
 *               000  (y-z)/4     sinpi(t)     cospi(t)    tanpi(t)
 *               001  (z+1-y)/4   cospi(t)     sinpi(t)    1/tanpi(t)
 *               010  (y-z)/4     cospi(t)    -sinpi(t)   -1/tanpi(t)
 *               011  (z+1-y)/4   sinpi(t)    -cospi(t)   -tanpi(t)
 *               100  (y-z)/4    -sinpi(t)    -cospi(t)    tanpi(t)
 *               101  (z+1-y)/4  -cospi(t)    -sinpi(t)    1/tanpi(t)
```



```
 *               110  (y-z)/4    -cospi(t)     sinpi(t)   -1/tanpi(t)
 *               111  (z+1-y)/4  -sinpi(t)     cospi(t)   -tanpi(t)
 *              ----------------------------------------------------
 *
 * NOTE. This program compute sinpi/cospi(t<0.25) by __k_sin/cos(pi*t, 0.0).
 * This will return a result with error slightly more than one ulp (but less
 * than 2 ulp). If one wants accurate result,  one may break up pi*t in
 * high (tpi_h) and low (tpi_l) parts and call __k_sin/cos(tip_h, tip_lo)
 * instead.
 */

#include "libm.h"
#include "longdouble.h"

#include <sys/isa_defs.h>

#define I(q, m)                         ((int *)&(q))[m]
#define U(q, m)                         ((unsigned *)&(q))[m]
#define I(q, m) ((int *) &(q))[m]
#define U(q, m) ((unsigned *) &(q))[m]
#if defined(__i386) || defined(__amd64)
#define LDBL_MOST_SIGNIF_I(ld)          ((I(ld, 2) << 16) | (0xffff & (I(ld, \
        1) >> 15)))
#define LDBL_MOST_SIGNIF_I(ld)  ((I(ld, 2) << 16) | (0xffff & (I(ld, 1) >> 15)))
#define LDBL_LEAST_SIGNIF_U(ld)         U(ld, 0)
#define PREC                            64
#define PRECM1                          63
#define PRECM2                          62

#endif /* ! codereview */
static const long double twoPRECM2 = 9.22337203685477580800000000000000e+18L;
#else
#define LDBL_MOST_SIGNIF_I(ld)          I(ld, 0)
#define LDBL_LEAST_SIGNIF_U(ld)         U(ld, sizeof (long double) / \
        sizeof (int) - 1)
#define LDBL_LEAST_SIGNIF_U(ld) U(ld, sizeof(long double) / sizeof(int) - 1)
#define PREC                            113
#define PRECM1                          112
#define PRECM2                          111

#endif /* ! codereview */
static const long double twoPRECM2 = 5.19229685853482762853049632922009 6e+33L;
#endif

static const long double zero = 0.0L,
        quater = 0.25L,
        one = 1.0L,
        pi = 3.14159265358979323846264338327950288419 7e+0000L,
        sqrth = 0.70710678118654752440084436210484903928483593768847 4,
        tiny = 1.0e-100;
static const long double
zero    = 0.0L,
quater  = 0.25L,
one     = 1.0L,
pi      = 3.14159265358979323846264338327950288419 7e+0000L,
sqrth   = 0.70710678118654752440084436210484903928483593768847 4,
tiny    = 1.0e-100;

void
sincospil(long double x, long double *s, long double *c)
{
sincospil(long double x, long double *s, long double *c) {
        long double y, z, t;
        int hx, n, k;
        unsigned lx;
```

```
116         hx = LDBL_MOST_SIGNIF_I(x);
117         lx = LDBL_LEAST_SIGNIF_U(x);
118         k = ((hx & 0x7fff0000) >> 16) - 0x3fff;

120 #endif /* ! codereview */
121         if (k >= PRECM2) {              /* |x| >= 2**(Prec-2) */
122                 if (k >= 16384) {
123                         *s = *c = x - x;
124                 } else {
 54                 }
 55                 else {
125                         if (k >= PREC) {
126                                 *s = zero;
127                                 *c = one;
128                         } else if (k == PRECM1) {
 59                         }
 60                         else if (k == PRECM1) {
129                                 if ((lx & 1) == 0) {
130                                         *s = zero;
131                                         *c = one;
132                                 } else {
 64                                 }
 65                                 else {
133                                         *s = -zero;
134                                         *c = -one;
135                                 }
136                         } else { /* k = Prec - 2 */
 69                         }
 70                         else {  /* k = Prec - 2 */
137                                 if ((lx & 1) == 0) {
138                                         *s = zero;
139                                         *c = one;
140                                 } else {
 74                                 }
 75                                 else {
141                                         *s = one;
142                                         *c = zero;
143                                 }

145 #endif /* ! codereview */
146                                 if ((lx & 2) != 0) {
147                                         *s = -*s;
148                                         *c = -*c;
149                                 }
150                         }
151                 }
152         } else if (k < -2) {            /* |x| < 0.25 */
 79         }
 80         else if (k < -2)        /* |x| < 0.25 */
153                 *s = __k_sincosl(pi * fabsl(x), zero, c);
154         } else {
 82         else {
155                 /* y = |4x|, z = floor(y), and n = (int)(z mod 8.0) */
156                 y = 4.0L * fabsl(x);

158 #endif /* ! codereview */
159                 if (k < PRECM2) {
160                         z = y + twoPRECM2;
161                         n = LDBL_LEAST_SIGNIF_U(z) & 7; /* 3 LSb of z */
162                         t = z - twoPRECM2;
163                         k = 0;

165                         if (t == y) {
 85                         if (t == y)
166                                 k = 1;
167                         } else if (t > y) {
```



```
 87                         else if (t > y) {
168                                 n -= 1;
169                                 t = quater + (y - t) * quater;
170                         } else {
 90                         }
 91                         else
171                                 t = (y - t) * quater;
172                         }
173                 } else { /* k = Prec-3 */
 94                 else {  /* k = Prec-3 */
174                         n = LDBL_LEAST_SIGNIF_U(y) & 7; /* 3 LSb of z */
175                         k = 1;
176                 }

178 #endif /* ! codereview */
179                 if (k) {                /* x = N/4 */
180                         if ((n & 1) != 0) {
 98                         if ((n & 1) != 0)
181                                 *s = *c = sqrth + tiny;
182                         } else if ((n & 2) == 0) {
100                         else
101                                 if ((n & 2) == 0) {
183                                 *s = zero;
184                                 *c = one;
185                         } else {
104                                 }
105                                 else {
186                                 *s = one;
187                                 *c = zero;
188                         }

190 #endif /* ! codereview */
191                         if ((n & 4) != 0)
192                                 *s = -*s;

194 #endif /* ! codereview */
195                         if (((n + 1) & 4) != 0)
196                                 *c = -*c;
197                 } else {
109                 }
110                 else {
198                         if ((n & 1) != 0)
199                                 t = quater - t;

201 #endif /* ! codereview */
202                         if (((n + (n & 1)) & 2) == 0)
203                                 *s = __k_sincosl(pi * t, zero, c);
204                         else
205                                 *c = __k_sincosl(pi * t, zero, s);

207 #endif /* ! codereview */
208                         if ((n & 4) != 0)
209                                 *s = -*s;

211 #endif /* ! codereview */
212                         if (((n + 2) & 4) != 0)
213                                 *c = -*c;
214                 }
215         }

217 #endif /* ! codereview */
218         if (hx < 0)
219                 *s = -*s;
220 }

222 #endif /* ! codereview */
```

```
#undef U
#undef LDBL_LEAST_SIGNIF_U
#undef I
#undef LDBL_MOST_SIGNIF_I
```

```
**********************************************************
    2294 Thu Jun  6 20:38:24 2019
new/usr/src/lib/libm/common/LD/sinhl.c
11210 libm should be cstyle(10NBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __sinhl = sinhl

#include "libm.h"
#include "longdouble.h"

/* BEGIN CSTYLED */
/*
 * SINH(X)
/* SINH(X)
 * RETURN THE HYPERBOLIC SINE OF X
 *
 * Method :
 *      1. reduce x to non-negative by SINH(-x) = - SINH(x).
 *      2.
 *
 *                                              EXPM1(x) + EXPM1(x)/(EXPM1(x)+1)
 *          0 <= x <= lnovft     : SINH(x) := ---------------------------------
 *                                                          2
 *
 *     lnovft <= x <  INF        : SINH(x) := EXP(x-MEP1*ln2)*2**ME
 *
 * here
 *      lnovft          logarithm of the overflow threshold
 *                      = MEP1*ln2 chopped to machine precision.
 *      ME              maximum exponent
 *      MEP1            maximum exponent plus 1
 *
 * Special cases:
 *      SINH(x) is x if x is +INF, -INF, or NaN.
 *      only SINH(0)=0 is exact for finite argument.
```



```
 *
 */
/* END CSTYLED */
#endif /* ! codereview */

static const long double C[] = {
        0.5L,
        1.0L,
        1.13565234062941439487914 9e+04L,
        7.00444768624254908785898 5e-16L
};

#define half            C[0]
#define one             C[1]
#define lnovft          C[2]
#define lnovlo          C[3]

long double
sinhl(long double x)
{
        long double r, t;

        if (!finitel(x))
                return (x + x);         /* x is INF or NaN */

#endif /* ! codereview */
        r = fabsl(x);

#endif /* ! codereview */
        if (r < lnovft) {
                t = expm1l(r);
                r = copysignl((t + t / (one + t)) * half, x);
        } else {
                r = copysignl(expl((r - lnovft) - lnovlo), x);
                r = scalbnl(r, 16383);
        }

#endif /* ! codereview */
        return (r);
}
```

```
**********************************************************
    2869 Thu Jun  6 20:38:24 2019
new/usr/src/lib/libm/common/LD/sinl.c
11210 libm should be cstyle(10NBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __sinl = sinl

/* BEGIN CSTYLED */
/*
 * sinl(x)
/* INDENT OFF */
/* sinl(x)
 * Table look-up algorithm by K.C. Ng, November, 1989.
 *
 * kernel function:
 *      __k_sinl                ... sin function on [-pi/4,pi/4]
 *      __k_cosl                ... cos function on [-pi/4,pi/4]
 *      __rem_pio2l     ... argument reduction routine
 *
 * Method.
 *      Let S and C denote the sin and cos respectively on [-PI/4, +PI/4].
 *      1. Assume the argument x is reduced to y1+y2 = x-k*pi/2 in
 *         [-pi/2 , +pi/2], and let n = k mod 4.
 *      2. Let S=S(y1+y2), C=C(y1+y2). Depending on n, we have
 *
 *          n        sin(x)      cos(x)        tan(x)
 *     ----------------------------------------------------------
 *          0          S           C             S/C
 *          1          C          -S            -C/S
 *          2         -S          -C             S/C
 *          3         -C           S            -C/S
 *     ----------------------------------------------------------
 *
 * Special cases:
 *      Let trig be any of sin, cos, or tan.
```



```
 *      trig(+-INF)  is NaN, with signals;
 *      trig(NaN)    is that NaN;
 *
 * Accuracy:
 *      computer TRIG(x) returns trig(x) nearly rounded.
 */
/* END CSTYLED */
/* INDENT ON */

#include "libm.h"
#include "longdouble.h"

#include <sys/isa_defs.h>

long double
sinl(long double x)
{
sinl(long double x) {
        long double y[2], z = 0.0L;
        int n, ix;

#endif /* ! codereview */
#if defined(__i386) || defined(__amd64)
        int *px = (int *)&x;
        int *px = (int *) &x;
#endif

        /* sin(Inf or NaN) is NaN */
        if (!finitel(x))
                return (x - x);
                return x - x;

        /* High word of x. */
#if defined(__i386) || defined(__amd64)
        XTOI(px, ix);
#else
        ix = *(int *)&x;
        ix = *(int *) &x;
#endif
        /* |x| ~< pi/4 */
        ix &= 0x7fffffff;
        if (ix <= 0x3ffe9220)
                return __k_sinl(x, z);

        if (ix <= 0x3ffe9220) {
                return (__k_sinl(x, z));
        }
#endif /* ! codereview */
        /* argument reduction needed */
        else {
                n = __rem_pio2l(x, y);

#endif /* ! codereview */
                switch (n & 3) {
                case 0:
                        return (__k_sinl(y[0], y[1]));
                        return __k_sinl(y[0], y[1]);
                case 1:
                        return (__k_cosl(y[0], y[1]));
                        return __k_cosl(y[0], y[1]);
                case 2:
                        return (-__k_sinl(y[0], y[1]));
                        return -__k_sinl(y[0], y[1]);
                case 3:
                        return (-__k_cosl(y[0], y[1]));
                        return -__k_cosl(y[0], y[1]);
```

```
                        /* NOTREACHED */
                }
        }

        return (0.0L);
        return 0.0L;
}
```

_____***unchanged_portion_omitted_***

```
**********************************************************
    5648 Thu Jun  6 20:38:25 2019
new/usr/src/lib/libm/common/LD/sinpil.c
11210 libm should be cstyle(10NBLD) clean
**********************************************************
   1 /*
   2  * CDDL HEADER START
   3  *
   4  * The contents of this file are subject to the terms of the
   5  * Common Development and Distribution License (the "License").
   6  * You may not use this file except in compliance with the License.
   7  *
   8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
   9  * or http://www.opensolaris.org/os/licensing.
  10  * See the License for the specific language governing permissions
  11  * and limitations under the License.
  12  *
  13  * When distributing Covered Code, include this CDDL HEADER in each
  14  * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  15  * If applicable, add the following below this CDDL HEADER, with the
  16  * fields enclosed by brackets "[]" replaced with your own identifying
  17  * information: Portions Copyright [yyyy] [name of copyright owner]
  18  *
  19  * CDDL HEADER END
  20  */

  22 /*
  23  * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
  24  */

  26 #endif /* ! codereview */
  27 /*
  28  * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
  29  * Use is subject to license terms.
  30  */

  32 /* BEGIN CSTYLED */
  33 /*
  34  * long double sinpil(long double x),
  25 /* long double sinpil(long double x),
  35  * return long double precision sinl(pi*x).
  36  *
  37  * Algorithm, 10/17/2002, K.C. Ng
  38  * ------------------------------
  39  * Let y = |4x|, z = floor(y), and n = (int)(z mod 8.0) (displayed in binary).
  40  *      1. If y == z, then x is a multiple of pi/4. Return the following values:
  41  *             ----------------------------------------------------
  42  *               n  x mod 2    sin(x*pi)    cos(x*pi)   tan(x*pi)
  43  *             ----------------------------------------------------
  44  *              000  0.00       +0 ___       +1 ___       +0
  45  *              001  0.25       +\/0.5       +\/0.5       +1
  46  *              010  0.50       +1 ___       +0 ___       +inf
  47  *              011  0.75       +\/0.5       -\/0.5       -1
  48  *              100  1.00       -0 ___       -1 ___       +0
  49  *              101  1.25       -\/0.5       -\/0.5       +1
  50  *              110  1.50       -1 ___       -0 ___       +inf
  51  *              111  1.75       -\/0.5       +\/0.5       -1
  52  *             ----------------------------------------------------
  53  *      2. Otherwise,
  54  *             ----------------------------------------------------
  55  *               n     t        sin(x*pi)    cos(x*pi)   tan(x*pi)
  56  *             ----------------------------------------------------
  57  *              000  (y-z)/4     sinpi(t)     cospi(t)    tanpi(t)
  58  *              001  (z+1-y)/4   cospi(t)     sinpi(t)    1/tanpi(t)
  59  *              010  (y-z)/4     cospi(t)    -sinpi(t)   -1/tanpi(t)
  60  *              011  (z+1-y)/4   sinpi(t)    -cospi(t)   -tanpi(t)
```



```
  61  *              100  (y-z)/4    -sinpi(t)    -cospi(t)    tanpi(t)
  62  *              101  (z+1-y)/4  -cospi(t)    -sinpi(t)    1/tanpi(t)
  63  *              110  (y-z)/4    -cospi(t)     sinpi(t)   -1/tanpi(t)
  64  *              111  (z+1-y)/4  -sinpi(t)     cospi(t)   -tanpi(t)
  65  *             ----------------------------------------------------
  66  *
  67  * NOTE. This program compute sinpi/cospi(t<0.25) by __k_sin/cos(pi*t, 0.0).
  68  * This will return a result with error slightly more than one ulp (but less
  69  * than 2 ulp). If one wants accurate result,  one may break up pi*t in
  70  * high (tpi_h) and low (tpi_l) parts and call __k_sin/cos(tip_h, tip_lo)
  71  * instead.
  72  */
  73 /* END CSTYLED */
  74 #endif /* ! codereview */

  76 #include "libm.h"
  77 #include "longdouble.h"

  79 #include <sys/isa_defs.h>

  81 #define I(q, m)                          ((int *)&(q))[m]
  82 #define U(q, m)                          ((unsigned *)&(q))[m]
  64 #define I(q, m) ((int *) &(q))[m]
  65 #define U(q, m) ((unsigned *) &(q))[m]
  83 #if defined(__i386) || defined(__amd64)
  84 #define LDBL_MOST_SIGNIF_I(ld)           ((I(ld, 2) << 16) | (0xffff & (I(ld, \
  85         1) >> 15)))
  67 #define LDBL_MOST_SIGNIF_I(ld)  ((I(ld, 2) << 16) | (0xffff & (I(ld, 1) >> 15)))
  86 #define LDBL_LEAST_SIGNIF_U(ld)          U(ld, 0)
  87 #define PREC                             64
  88 #define PRECM1                           63
  89 #define PRECM2                           62

  91 #endif /* ! codereview */
  92 static const long double twoPRECM2 = 9.223372036854775808000000000000000e+18L;
  93 #else
  94 #define LDBL_MOST_SIGNIF_I(ld)           I(ld, 0)
  95 #define LDBL_LEAST_SIGNIF_U(ld)          U(ld, sizeof (long double) / \
  96         sizeof (int) - 1)
  72 #define LDBL_LEAST_SIGNIF_U(ld) U(ld, sizeof(long double) / sizeof(int) - 1)
  97 #define PREC                             113
  98 #define PRECM1                           112
  99 #define PRECM2                           111

 101 #endif /* ! codereview */
 102 static const long double twoPRECM2 = 5.192296858534827628530496329220096e+33L;
 103 #endif

 105 static const long double zero = 0.0L,
 106         quater = 0.25L,
 107         one = 1.0L,
 108         pi = 3.141592653589793238462643383279502884197e+0000L,
 109         sqrth = 0.707106781186547524400844362104849039284835937688474,
 110         tiny = 1.0e-100;
  76 static const long double
  77 zero    = 0.0L,
  78 quater  = 0.25L,
  79 one     = 1.0L,
  80 pi      = 3.141592653589793238462643383279502884197e+0000L,
  81 sqrth   = 0.707106781186547524400844362104849039284835937688474,
  82 tiny    = 1.0e-100;

 112 long double
 113 sinpil(long double x)
 114 {
  85 sinpil(long double x) {
```

```
115         long double y, z, t;
116         int hx, n, k;
117         unsigned lx;

119         hx = LDBL_MOST_SIGNIF_I(x);
120         lx = LDBL_LEAST_SIGNIF_U(x);
121         k = ((hx & 0x7fff0000) >> 16) - 0x3fff;

123 #endif /* ! codereview */
124         if (k >= PRECM2) {              /* |x| >= 2**(Prec-2) */
125                 if (k >= 16384) {
 93                 if (k >= 16384)
126                         y = x - x;
127                 } else {
128                         if (k >= PREC) {
 95                 else {
 96                         if (k >= PREC)
129                                 y = zero;
130                         } else if (k == PRECM1) {
131                                 y = (lx & 1) == 0 ? zero : -zero;
132                         } else { /* k = Prec - 2 */
 98                         else if (k == PRECM1)
 99                                 y = (lx & 1) == 0 ? zero: -zero;
100                         else {  /* k = Prec - 2 */
133                                 y = (lx & 1) == 0 ? zero : one;

135 #endif /* ! codereview */
136                                 if ((lx & 2) != 0)
137                                         y = -y;
138                         }
139                 }
140         } else if (k < -2) {            /* |x| < 0.25 */
102         }
103         else if (k < -2)        /* |x| < 0.25 */
141                 y = __k_sinl(pi * fabsl(x), zero);
142         } else {
105         else {
143                 /* y = |4x|, z = floor(y), and n = (int)(z mod 8.0) */
144                 y = 4.0L * fabsl(x);

146 #endif /* ! codereview */
147                 if (k < PRECM2) {
148                         z = y + twoPRECM2;
149                         n = LDBL_LEAST_SIGNIF_U(z) & 7; /* 3 LSb of z */
150                         t = z - twoPRECM2;
151                         k = 0;

153                         if (t == y) {
108                         if (t == y)
154                                 k = 1;
155                         } else if (t > y) {
110                         else if (t > y) {
156                                 n -= 1;
157                                 t = quater + (y - t) * quater;
158                         } else {
113                         }
114                         else
159                                 t = (y - t) * quater;
160                         }
161                 } else { /* k = Prec-3 */
117                 else {  /* k = Prec-3 */
162                         n = LDBL_LEAST_SIGNIF_U(y) & 7; /* 3 LSb of z */
163                         k = 1;
164                 }

166 #endif /* ! codereview */
```



```
167                 if (k) {                /* x = N/4 */
168                         if ((n & 1) != 0)
169                                 y = sqrth + tiny;
170                         else
171                                 y = (n & 2) == 0 ? zero : one;

173 #endif /* ! codereview */
174                         if ((n & 4) != 0)
175                                 y = -y;
176                 } else {
121                 }
122                 else {
177                         if ((n & 1) != 0)
178                                 t = quater - t;

180 #endif /* ! codereview */
181                         if (((n + (n & 1)) & 2) == 0)
182                                 y = __k_sinl(pi * t, zero);
183                         else
184                                 y = __k_cosl(pi * t, zero);

186 #endif /* ! codereview */
187                         if ((n & 4) != 0)
188                                 y = -y;
189                 }
190         }

192         return (hx >= 0 ? y : -y);
125         return hx >= 0 ? y : -y;
193 }

195 #endif /* ! codereview */
196 #undef U
197 #undef LDBL_LEAST_SIGNIF_U
198 #undef I
199 #undef LDBL_MOST_SIGNIF_I
```

```
**********************************************************
    2644 Thu Jun  6 20:38:25 2019
new/usr/src/lib/libm/common/LD/tanhl.c
11210 libm should be cstyle(10NBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __tanhl = tanhl

/*
 * tanhl(x) returns the hyperbolic tangent of x
 *
 * Method :
 *      1. reduce x to non-negative:  tanhl(-x) = - tanhl(x).
 *      2.
 *        0      <  x <=  small    :  tanhl(x) := x
 *                                                 -expm1l(-2x)
 *        small  <  x <=  1        :  tanhl(x) := --------------
 *                                                expm1l(-2x) + 2
 *                                                       2
 *        1      <= x <= threshold :  tanhl(x) := 1 -  ---------------
 *                                                   expm1l(2x) + 2
 *     threshold <  x <= INF       :  tanhl(x) := 1.
 *
 * where
 *      single :        small = 1.e-5           threshold = 11.0
 *      double :        small = 1.e-10          threshold = 22.0
 *      quad   :        small = 1.e-20          threshold = 45.0
 *
 * Note: threshold was chosen so that
 *              fl(1.0+2/(expm1(2*threshold)+2)) == 1.
 *
 * Special cases:
 *      tanhl(NaN) is NaN;
 *      only tanhl(0.0)=0.0 is exact for finite argument.
 */
```



```
#include "libm.h"
#include "longdouble.h"

static const long double small = 1.0e-20L,
        one = 1.0,
        two = 2.0,
static const long double small = 1.0e-20L, one = 1.0, two = 2.0,
#ifndef lint
        big = 1.0e+20L,
#endif
        threshold = 45.0L;

long double
tanhl(long double x)
{
tanhl(long double x) {
        long double t, y, z;
        int signx;

#endif /* ! codereview */
#ifndef lint
        volatile long double dummy __unused;
#endif

        if (isnanl(x))
                return (x + x);         /* x is NaN */

#endif /* ! codereview */
        signx = signbitl(x);
        t = fabsl(x);
        z = one;

#endif /* ! codereview */
        if (t <= threshold) {
                if (t > one) {
                if (t > one)
                        z = one - two / (expm1l(t + t) + two);
                } else if (t > small) {
                else if (t > small) {
                        y = expm1l(-t - t);
                        z = -y / (y + two);
                } else {
#ifndef lint
                        dummy = t + big;
                        /* inexact if t != 0 */
#endif
                        return (x);
                }
        } else if (!finitel(t)) {
        } else if (!finitel(t))
                return (copysignl(one, x));
        } else {
        else
                return (signx ? -z + small * small : z - small * small);
        }

#endif /* ! codereview */
        return (signx ? -z : z);
}
```

```
**********************************************************
    2632 Thu Jun  6 20:38:26 2019
new/usr/src/lib/libm/common/LD/tanl.c
11210 libm should be cstyle(10NBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __tanl = tanl

/* BEGIN CSTYLED */
/*
 * cosl(x)
/* INDENT OFF */
/* cosl(x)
 * Table look-up algorithm by K.C. Ng, November, 1989.
 *
 * kernel function:
 *      __k_tanl        ... tangent function on [-pi/4,pi/4]
 *      __rem_pio2l     ... argument reduction routine
 *
 * Method.
 *      Let S and C denote the sin and cos respectively on [-PI/4, +PI/4].
 *      1. Assume the argument x is reduced to y1+y2 = x-k*pi/2 in
 *         [-pi/2 , +pi/2], and let n = k mod 4.
 *      2. Let S=S(y1+y2), C=C(y1+y2). Depending on n, we have
 *
 *          n        sin(x)      cos(x)        tan(x)
 *     ----------------------------------------------------------
 *          0          S           C             S/C
 *          1          C          -S            -C/S
 *          2         -S          -C             S/C
 *          3         -C           S            -C/S
 *     ----------------------------------------------------------
 *
 * Special cases:
 *      Let trig be any of sin, cos, or tan.
 *      trig(+-INF)  is NaN, with signals;
```



```
 *      trig(NaN)    is that NaN;
 *
 * Accuracy:
 *      computer TRIG(x) returns trig(x) nearly rounded.
 */
/* END CSTYLED */
/* INDENT ON */

#include "libm.h"
#include "longdouble.h"

#include <sys/isa_defs.h>

long double
tanl(long double x)
{
tanl(long double x) {
        long double y[2], z = 0.0L;
        int n, ix;

#endif /* ! codereview */
#if defined(__i386) || defined(__amd64)
        int *px = (int *)&x;
        int *px = (int *) &x;
#endif

        /* trig(Inf or NaN) is NaN */
        if (!finitel(x))
                return (x - x);
                return x - x;

        /* High word of x. */
#if defined(__i386) || defined(__amd64)
        XTOI(px, ix);
#else
        ix = *(int *)&x;
        ix = *(int *) &x;
#endif

        /* |x| ~< pi/4 */
        ix &= 0x7fffffff;
        if (ix <= 0x3ffe9220)
                return __k_tanl(x, z, 0);

        if (ix <= 0x3ffe9220) {
                return (__k_tanl(x, z, 0));
        }
#endif /* ! codereview */
        /* argument reduction needed */
        else {
                n = __rem_pio2l(x, y);
                return (__k_tanl(y[0], y[1], n & 1));
                return __k_tanl(y[0], y[1], n & 1);
        }
}
______unchanged_portion_omitted_
```

```
**********************************************************
   11258 Thu Jun  6 20:38:26 2019
new/usr/src/lib/libm/common/Q/_TBL_atanl.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

/*
 * Table of constants for atanl.
 * By K.C. Ng, March 9, 1989
 */

#include "libm.h"

const long double _TBL_atanl_hi[] = {
        +1.2435499454676143503135484916387102416 57e-0001L,
        +1.3203976161463874927468440652656953226 25e-0001L,
        +1.3970887428916364518336777673909505681 61e-0001L,
        +1.4736148108865163560980276039684551821 07e-0001L,
        +1.5499674192394098230371437493349219133 37e-0001L,
        +1.6261382859794857537364156376155780062 02e-0001L,
        +1.7021192528547440449049660709976171369 54e-0001L,
        +1.7779022899267607079662479921582468899 46e-0001L,
        +1.8534794999569476488602596122854464667 26e-0001L,
        +1.9288431225797466419705871069022730349 88e-0001L,
        +2.0039855382587851465394578503437838446 15e-0001L,
        +2.0788992720226299360533498310299432475 63e-0001L,
        +2.1535769969773804802445962716648964165 74e-0001L,
        +2.2280115375939451577103212214043255525 02e-0001L,
        +2.3021958727684373024017095967980299065 55e-0001L,
        +2.3761231386547125247388363432563777919 89e-0001L,
        +2.4497866312686415417208248121127580641 96e-0001L,
        +2.5962962940825753102994644318397190560 11e-0001L,
        +2.7416745111965879759937189834217578592 44e-0001L,
        +2.8858736189407739562361141995821834504 33e-0001L,
        +3.0288486837497140556055609450555821812 28e-0001L,
        +3.1705575320914700980901557667446732975 85e-0001L,
        +3.3109607670413209494433878775694455421 26e-0001L,
        +3.4500217720710510886768128690005168408 29e-0001L,
        +3.5877067027057222039592006392646052215 36e-0001L,
        +3.7239844667675422192365503828370182641 41e-0001L,
        +3.8588266939807377589769548460723139638 19e-0001L,
        +3.9922076957525256561471669615886476491 10e-0001L,
        +4.1241044159738730689979128966712694260 92e-0001L,
        +4.2544963737004228954226360518079233013 82e-0001L,
        +4.3833655985795780544561604921477130895 88e-0001L,
        +4.5106965598852347637563925728219344073 80e-0001L,
        +4.6364760900080611621425623146121439713 34e-0001L,
        +4.8833395105640552386716496074706484459 64e-0001L,
        +5.1238946031073770666660102058425923805 56e-0001L,
        +5.3581123796046370026908506870769144698 47e-0001L,
        +5.5859931534356243597150821640166122875 87e-0001L,
        +5.8075635356767039920327447500150082375 12e-0001L,
        +6.0228734613496418168212269420423291922 46e-0001L,
        +6.2319932993406593099247534906037459367 79e-0001L,
        +6.4350110879328438680280922871732260447 27e-0001L,
        +6.6320299270609325536325431023827583417 23e-0001L,
        +6.8231655487474807825642998171115298784 73e-0001L,
        +7.0085440788445017245795128178675127318 62e-0001L,
        +7.1882999962162450541701415152590469891 04e-0001L,
        +7.3625742898142813174283527108914662479 27e-0001L,
        +7.5315128096219438952473937026902888600 57e-0001L,
        +7.6952648040565826040682003598565401726 60e-0001L,
        +7.8539816339744830961566084581987569936 98e-0001L,
        +8.1569192331622341102146083874564582672 28e-0001L,
        +8.4415398611317100251784414827164746736 83e-0001L,
        +8.7090345707565295314017311259781407291 65e-0001L,
        +8.9605538457134395617480071802993779546 60e-0001L,
        +9.1971960535041681722860345482108940969 31e-0001L,
        +9.4200004037946366473793717053459362115 89e-0001L,
        +9.6299433068093620181519583599709989677 30e-0001L,
        +9.8279372324732906798571061101466603762 57e-0001L,
        +1.0014831356942347329183295953014374896 34e+0000L,
        +1.0191413442663497346383429170230636212 35e+0000L,
        +1.0358412530088001765846944703254440735 48e+0000L,
        +1.0516502125483736674598673120862999026 92e+0000L,
        +1.0666303653157435630791763474202799086 01e+0000L,
        +1.0808390005411683108871567292171997859 00e+0000L,
        +1.0943289073211899198927883146102352763 03e+0000L,
        +1.1071487177940905030170654601785370497 54e+0000L,
        +1.1309537439791604464709335155363277560 03e+0000L,
        +1.1525719972156675180401498626127514672 83e+0000L,
        +1.1722738811284763866005949441337046006 86e+0000L,
        +1.1902899496825317329277337748293182803 38e+0000L,
        +1.2068173702852525303955115800565576626 68e+0000L,
        +1.2220253232109896370417417439225704120 29e+0000L,
        +1.2360594894780819419094519711090786146 21e+0000L,
        +1.2490457723982544258299170772810900483 55e+0000L,
        +1.2610933822524404193139408812473357640 12e+0000L,
        +1.2722973952087173412961937498224805746 46e+0000L,
        +1.2827408797442707473628852511364955164 07e+0000L,
        +1.2924966677897852679030914214070816723 53e+0000L,
        +1.3016288340091961438047858503666855024 45e+0000L,
        +1.3101939350475556342564376891719053437 54e+0000L,
        +1.3182420510168370498593302023271363040 43e+0000L,
        +1.3258176636680324650592392104284756886 16e+0000L,
        +1.3397056595989995393283037525895557850 24e+0000L,
        +1.3521273809209546571891479413898127598 77e+0000L,
        +1.3633001003596939542892985278250991560 27e+0000L,
        +1.3734007669450158608612719264449610604 84e+0000L,
        +1.3825748214901258580599674177685681639 5e+0000L,
        +1.3909428270024183486427686943836432395 49e+0000L,
        +1.3986055122719575950126700816114282727 86e+0000L,
```

```
        +1.40564764938026978095219340199580806644 1e+0000L,
        +1.41214106460849521536761367185489085282e+0000L,
        +1.41814699839963145940386030397009886326 1e+0000L,
        +1.42371797140649411890181904661072971089 0e+0000L,
        +1.42889927219073269641847007453719840013 9e+0000L,
        +1.43373015248470898664047190966988738802 6e+0000L,
        +1.43824479449822259796140424793548160396 7e+0000L,
        +1.44247309910910182002529205993772918103 5e+0000L,
        +1.44644133224813518419996684247588038661 1e+0000L,
};

const long double _TBL_atanl_lo[] = {
        +1.40748691976280638023172028204143085906 5e-0036L,
        -4.95969615947399255557304394379996694998 7e-0036L,
        +8.95277456251946488739312134463618378815 2e-0036L,
        +1.18804374232078957181807658435449644303 0e-0035L,
        -2.78102781120451453784253751282343545146 3e-0037L,
        +1.47972203770238003272955362343151472623 9e-0036L,
        -4.21695614005481987328703848018496340681 9e-0036L,
        +7.24312296669134846499303236563160234946 8e-0036L,
        -2.15734300898391702998956793537906515911 9e-0036L,
        -9.95157454051267235544523672981286051863 1e-0036L,
        -3.90655589923248381816175697303978765674 3e-0036L,
        +5.52602922717937268132119806646611303144 4e-0036L,
        +8.84157222159143218076822543180364520436 9e-0036L,
        -8.17677287915861792541933236282855882094 4e-0036L,
        -1.33441230346561422437971138230283387642 1e-0036L,
        -4.49273312078133829089307339246813258921 9e-0036L,
        +4.49455114718124903932018243367625014833 6e-0036L,
        -1.66880815042792235557767244596484405672 7e-0035L,
        +1.56297575861079557694610865689373296841 1e-0035L,
        -2.23898355633080785525079703853315108481 1e-0035L,
        -4.83123217455473115518704506711821629583 2e-0036L,
        -1.43361723529058328769589266109806988443 1e-0035L,
        -8.74401819988999328029891741709605817248 1e-0036L,
        +5.92846360085298374457803607854645559386 5e-0036L,
        -2.23766512484362412760610552950435149936 3e-0035L,
        +6.07458375993361054142803107566774421364 8e-0036L,
        +1.53721871104519496777923447620299670230 9e-0035L,
        +2.09760680567511562416571215824787902471 6e-0035L,
        -5.56239564054954380607268622026228191149 7e-0036L,
        +1.96973667078324718418584119348973519015 2e-0035L,
        +2.10703119644794885090347336394248875437 0e-0035L,
        -2.30273563629820016022565185108542298445 6e-0035L,
        +4.89509642257333492668618435220297705684 8e-0036L,
        -7.23801434777944582138727230508202647576 6e-0036L,
        +1.63656488657036140316374433960495688581 1e-0035L,
        -3.98858119582345307937291299198032341974 0e-0035L,
        +4.15877221209126135104177839232274259734 4e-0035L,
        +3.83474214545564721536846873773371350273 9e-0035L,
        -9.22511789336387217235158964654889909065 9e-0036L,
        +1.40946196904559895261757367418546503965 4e-0036L,
        +3.35688578054722352706128514258108036794 5e-0035L,
        +3.90909910555225523950181068032321188034 0e-0035L,
        +5.29564169796542081405218627072970396535 9e-0036L,
        -5.09608468199455143678470639236625071367 2e-0036L,
        -4.49590144252776158583296803939183152050 0e-0035L,
        +3.80392265445516342665668576159626142903 4e-0035L,
        -4.40565228728955121083086421966116896576 2e-0036L,
        +1.60250241924821610762238077534256190769 5e-0036L,
        +2.16795253253094525619926100651083806352 6e-0035L,
        +1.98440380135154221257153629257367541040 7e-0035L,
        +3.91396194717997468345052273535684324572 4e-0035L,
        +2.11134438079754535055184534367995618547 3e-0035L,
        +3.15585727744469275503981694439277018543 e-0035L,
        +1.62950445203554614082655856195002383356 1e-0035L,
```



```
        -3.50872452092703058561512303561712089458 0e-0035L,
        +2.90410418642828556795910552709461173000 9e-0035L,
        -2.31288434538183565909319952098066272332 8e-0035L,
        -7.71249231814715784399679738207159798748 1e-0035L,
        +2.75390278298869224290920635904499538193 3e-0035L,
        -9.45008994531813089510845459908375277344 5e-0035L,
        -7.30617553020320923375949460016431815910 1e-0035L,
        -4.17361448139537521939527701574043190664 3e-0035L,
        +3.43699483562564070453448552628642574964 7e-0035L,
        -6.37902434922980909073020849242756348974 8e-0035L,
        -9.68429438163532612910041278660940048846 4e-0036L,
        +4.87467575391388709092759583266980605772 8e-0035L,
        -8.75338864770841908845116013685854785275 1e-0035L,
        +1.42847439923279188926925511380822048416 0e-0035L,
        +5.72627762110733895425656256934744905722 8e-0035L,
        -3.22548831487804112455948222707503549131 7e-0035L,
        +7.88535481906098773259655252523767351356 1e-0035L,
        +8.40817367390371940975150383653688292831 8e-0035L,
        +7.47228703575636838150782429819345423942 5e-0035L,
        +7.99772028257934352894348136008700704397 4e-0036L,
        -8.05778407733621390548484923462939533215 3e-0035L,
        +1.42177467536705830654900402090530858042 6e-0035L,
        +1.22324869142212050041097435603231269932 7e-0035L,
        +8.96960550708300364473619572179466404214 6e-0035L,
        -3.14803944350818844106860667398493670460 9e-0035L,
        -5.09271460407153450132406425176115711623 6e-0035L,
        -5.74319977159241365681338594327058588616 6e-0035L,
        -4.39204514050837702790997660804795084430 0e-0035L,
        +9.11067539849077155630186667763132196437 2e-0035L,
        -3.70325690142728410095124007730643565350 3e-0035L,
        +8.81674194297467142769098254051317691390 7e-0035L,
        -3.83893416960283525037523128617033105192 3e-0036L,
        -3.34629593419608915463408955080142512133 5e-0035L,
        -3.92126267767860743839161884989555550809 9e-0035L,
        -7.83403973963778672558644945685914177502 2e-0035L,
        +7.46810186324569865206006403406243610055 8e-0035L,
        +8.91109186189569184511355948761654817983 9e-0035L,
        +3.94181606322718905304317971456687085749 1e-0035L,
        -4.10481140885801048201934356383271816121 9e-0035L,
        -2.31654194515821533263839447562209445011 5e-0035L,
        -1.84283125815253194093993302037054598200 7e-0035L,
        +7.14773165467094823454117120179094021222 0e-0035L,
        +2.99145015784358746621536377070195345257 1e-0035L,
};
```

```
**********************************************************
    9105 Thu Jun  6 20:38:27 2019
new/usr/src/lib/libm/common/Q/_TBL_cosl.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

/*
 * table of cosl(x) where x is 0.15625 + i*0.03125, i=0,1,...,74.
 * {0x3ffc4000,0,0,0} --> (inc 0x800) -->  {0x3ffe9000,0,0,0}
 *    0.15625              0.03125           0.78125  (pi/4 = 0.785395663...)
 */

#include "libm.h"

const long double _TBL_cosl_hi[] = {
        +9.8781778381647194410050303436321131650 93e-0001L,
        +9.8720237785483049039608853351162244439 52e-0001L,
        +9.8657190839949758875733740749530840860 15e-0001L,
        +9.8592638507066143574705925286943544415 08e-0001L,
        +9.8526581771821381620429470975957899393 59e-0001L,
        +9.8459021642159980601439510778206873644 41e-0001L,
        +9.8389959148966397217830935141648724533 67e-0001L,
        +9.8319395346049307252787576129896842751 34e-0001L,
        +9.8247331310125525748732768324362249501 47e-0001L,
        +9.8173768140803577633459614790470900319 71e-0001L,
        +9.8098706905669190469329896435309664556 9e-0001L,
        +9.8022148914756809624785186742174201828 72e-0001L,
        +9.7944095171554835999853095450298749332 58e-0001L,
        +9.7864546921965086788426767974327528425 34e-0001L,
        +9.7783505379795979333197157294445454933 30e-0001L,
        +9.7700971781641738480204569021457678846 24e-0001L,
        +9.7616947386863527672398903543513553369 67e-0001L,
        +9.7531433477570232647727985562226109386 01e-0001L,
        +9.7444431358598898034971105604543434405 01e-0001L,
        +9.7355942357494817145831251450989810120 12e-0001L,
        +9.7265967824491275267091305826756525978 51e-0001L,
```



```
        +9.7174509132488946761926649413250296432 11e-0001L,
        +9.7081567677034946294749054578504602702 55e-0001L,
        +9.6987144876301534499234404591693077612 67e-0001L,
        +9.6891242171064478414459544949418920534 05e-0001L,
        +9.6695002923067782200834162361053150340 50e-0001L,
        +9.6492861910477100958107466531574837140 01e-0001L,
        +9.6284831470937969989970109348021436468 62e-0001L,
        +9.6070924301556190306665935058131347170 46e-0001L,
        +9.5851153458122862730196940815491982178 56e-0001L,
        +9.5625532354317529697559994226302836116 90e-0001L,
        +9.5394074760889473398132479598761162283 19e-0001L,
        +9.5156794804817220214548821736427096216 57e-0001L,
        +9.4913706968446302766584742176210562300 77e-0001L,
        +9.4664826088605332184609950729553297611 08e-0001L,
        +9.4410167355700434563001769125312485996 00e-0001L,
        +9.4149746312788106864451123605367081465 37e-0001L,
        +9.3883578854626548863257830598471255415 86e-0001L,
        +9.3611681226705529029423741101950858803 18e-0001L,
        +9.3334070024254843565529922946999552659 09e-0001L,
        +9.3050762191231429114947679222955548064 11e-0001L,
        +9.2761775019285190962803079879996135019 18e-0001L,
        +9.2467126146703609850211301456013877099 96e-0001L,
        +9.2166833557335191817541136820271271423 83e-0001L,
        +9.1860915579491826783782497771854986258 01e-0001L,
        +9.1549390884830122856391773218022188166 45e-0001L,
        +9.1232278487211784649202954204734173375 77e-0001L,
        +9.0909597741543105165038173568447641749 05e-0001L,
        +9.0581368342593642074451666065270025770 88e-0001L,
        +9.0247610323794150492518327267589599949 48e-0001L,
        +8.9908344056013845621654492920937930653 80e-0001L,
        +8.9563590246317069890057000044625635034 48e-0001L,
        +8.9213369936699440472390025372378857507 67e-0001L,
        +8.8857704502803554331760902311602098009 73e-0001L,
        +8.8496615652614329169729653696664792642 36e-0001L,
        +8.8130125425134059914016190829810017288 13e-0001L,
        +8.7758256189037271611628158260382968094 01e-0001L,
        +8.6998471805841738882891559990146624298 87e-0001L,
        +8.6217447993488050436716251025332427412 50e-0001L,
        +8.5415375427738538514345178510510317644 12e-0001L,
        +8.4592449923106795445972307859749326242 46e-0001L,
        +8.3748872385052368531535334891724061715 13e-0001L,
        +8.2884848760932573481017179011911663815 10e-0001L,
        +8.2000589989723400825555063387655604252 68e-0001L,
        +8.1096311950521790218953480394108072435 20e-0001L,
        +8.0172235409841845060749260565296420782 77e-0001L,
        +7.9228585967717854314150132378170939853 02e-0001L,
        +7.8265594002627279693078744742813902594 85e-0001L,
        +7.7283494615247154481085184591342517756 39e-0001L,
        +7.6282527571057625050709875362542979186 21e-0001L,
        +7.5262937241806647605454132484714311591 93e-0001L,
        +7.4224972545850130699134725344961053672 06e-0001L,
        +7.3168886887382088631183875300008452901 50e-0001L,
        +7.2094938094569641804381278414844767909 2e-0001L,
        +7.1003388356607967497412164395949021943 33e-0001L,
};

const long double _TBL_cosl_lo[] = {
        +4.7427130783670589789246810762052641836 48e-0035L,
        -3.4009225800381533529090342076771815600 93e-0035L,
        -2.4732794993698536247625240121272072463 23e-0035L,
        -3.9023208770045180007162320645462385787 34e-0035L,
        +2.2656802950581806614151749777852792117 3e-0035L,
        -2.2547722464442032591705883021046629910 85e-0036L,
        +2.7341431894806620781048633072376126487 80e-0035L,
        -3.7019125606934464386562021684463556778 22e-0035L,
        -1.6492435889155758462546386801423034223 20e-0035L,
```

```
        +2.725042655698714891044457001868653187367e-0035L,
        -1.908992594100964198869963315362783449712e-0036L,
        -1.465547554627127716918860559012698704471e-0035L,
        +4.428780565915607570668447972900679899952e-0035L,
        +1.439313657623768907227720140857454695843e-0035L,
        -3.792074229051804169372108537791927020038e-0035L,
        -2.610779485320152706286660129045188117210e-0036L,
        -2.877279742494815830479448606269854599891e-0035L,
        +3.991065835589256680020290949615723238476e-0035L,
        +3.099479059550534193045145385925483327991e-0035L,
        +1.146611686911982702287167679510021879695e-0035L,
        -3.917592318193149049660769585602527582231e-0035L,
        -1.951971321999985008371800682574139933978e-0035L,
        +2.974588209723938591252776820212028367960e-0035L,
        -2.038390756570426530537115267786908745116e-0036L,
        -5.536347061134619893988732877493263844943e-0036L,
        -4.389722144327924120620880599904805370946e-0035L,
        -3.666858326708207750024755456027611364938e-0035L,
        +4.889869663833434507994220130518213362272e-0036L,
        -5.870115582315839607120133516012219562069e-0036L,
        +2.507707793716364811457350893931543805685e-0035L,
        +3.216165721908659970511036451358372071749e-0035L,
        +2.880756890524786020083959729246571876109e-0035L,
        +6.368426285981156583087492887998846060579e-0036L,
        +6.844339659916371522503091904688601360028e-0037L,
        -4.329063396630008909415294204988246215817e-0035L,
        +1.038125352401202296098224611721455839121e-0035L,
        +3.207093666031656020715902410548849578474e-0036L,
        -3.987580687739740313485850727522454807713e-0035L,
        +3.404815912367106584354098624390321615909e-0035L,
        -4.752557072516798311248008988313821999362e-0035L,
        +2.745410885517329825733352856854160918801e-0035L,
        +7.585203719163457562812011671268547121453e-0036L,
        -4.141871248600318251086493472511758380472e-0035L,
        -1.835879954339576229487102635414793218992e-0035L,
        +2.976082827782744334600577457984098492775e-0035L,
        -3.507755179553069548150909011683056358498e-0035L,
        +7.869038865563736742679481321788455681309e-0036L,
        +1.208860140284441557337760250856779527931e-0035L,
        -3.609503076059411697756765630044671398302e-0035L,
        +2.262828995013444190183062956802106020046e-0035L,
        -2.067726154909043706666702751547519756391e-0035L,
        +3.735937416598668830886204955423117851511e-0035L,
        -1.107719376025673147326930792646924920884e-0035L,
        +4.123542789546647314438136551770221119198e-0036L,
        +4.533705702883256304420378263134621416396e-0035L,
        -1.434191923121166877839456190096294453634e-0035L,
        -2.894849601813639248551925385406988512004e-0035L,
        -4.681686383005756267827413197921838600437e-0035L,
        -3.715568183175335822345624718357717998947e-0035L,
        -1.687075340130951528732220617225731715663e-0035L,
        +1.980549471419898781791643429252740528544e-0035L,
        +2.727619978720845330457771867732615590081e-0035L,
        +1.430825081004965817190481755062397701422e-0035L,
        -1.720088119552308234167243322979912469421e-0035L,
        +1.104812928567944364260514024188043464704e-0035L,
        +6.094878513052330893256279394589637408556e-0036L,
        +2.475195582284731678792488916738076213891e-0035L,
        +1.693320456792379194278077712885062541662e-0035L,
        +3.949752293412116642372415347411469162440e-0035L,
        +4.220674118886015050047489393823250795070e-0035L,
        +3.713069586576631896654508643111045710544e-0035L,
        -3.789252700498009135399234738712875263543e-0035L,
        +1.482556375489316971849917102931986196306e-0035L,
        +4.786912857336733794995363260508118324272e-0035L,
        -4.096232247636924432208967529079024417475e-0035L,
```



```
};
```

```
************************************************************
    4495 Thu Jun  6 20:38:27 2019
new/usr/src/lib/libm/common/Q/_TBL_expl.c
11210 libm should be cstyle(1ONBLD) clean
************************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

/*
 * Table of constants for expl.
 * By K.C. Ng, March 9, 1989
 */

#include "libm.h"

const long double _TBL_expl_hi[] = {
        +1.0000000000000000000000000000000000000000e+0000L,
        +1.0218971486541166782344801347832994397 82e+0000L,
        +1.0442737824274138403219664787399290087 85e+0000L,
        +1.0671404006768236181695211209928091626 07e+0000L,
        +1.0905077326652576592070106557607079789 93e+0000L,
        +1.1143867425958925363088129569196030678 00e+0000L,
        +1.1387886347566916537038302838415112547 20e+0000L,
        +1.1637248587775751381357359909218531234 3e+0000L,
        +1.1892071150027210667174999705604759152 93e+0000L,
        +1.2152473599804688781165202513387984576 24e+0000L,
        +1.2418578120734840485936774687265956055 11e+0000L,
        +1.2690509571917332225544190810323380047 15e+0000L,
        +1.2968395546510096659337541177924511598 35e+0000L,
        +1.3252366431597412946295370954987216741 13e+0000L,
        +1.3542555469368927282980147401407028043 43e+0000L,
        +1.3839098819638319548726595272651928180 2e+0000L,
        +1.4142135623730950488016887242096980785 70e+0000L,
        +1.4451808069770466200370062414716709056 78e+0000L,
        +1.4768261459394993113869074803740499239 24e+0000L,
        +1.5091644275934227397660195510331935314 20e+0000L,
        +1.5422108254079408236122918620907348413 07e+0000L,
        +1.5759808451078864864552701601819050089 06e+0000L,
```



```
        +1.6104903319492543081795206673574005834 59e+0000L,
        +1.6457554781539648445187567247258224456 67e+0000L,
        +1.6817928305074290860622509524664297900 80e+0000L,
        +1.7186192981224779156293443764563125045 16e+0000L,
        +1.7562521603732994831121606193753132212 94e+0000L,
        +1.7947090750031071864277032421277818143 54e+0000L,
        +1.8340080864093424634870831895882888560 77e+0000L,
        +1.8741676341102999013299989499544465344 39e+0000L,
        +1.9152065613971472938726112702958308878 50e+0000L,
        +1.9571441241754002690183222516268714911 90e+0000L,
};

const long double _TBL_expl_lo[] = {
        +0.0000000000000000000000000000000000000000e+0000L,
        +1.8050678742033095474557333305457378646 51e-0035L,
        -9.3745202922804274219575674197308321438 43e-0035L,
        -1.5969684472927587707129096302314999712 33e-0035L,
        +9.1124934101250229785116861016724866621 19e-0035L,
        -6.5042282069785482872303747752593887109 85e-0035L,
        -8.1484688445258511373256917674881553236 05e-0035L,
        -5.0662145767218003133723307451429033509 63e-0035L,
        -1.3598309746888169737498756382459191189 24e-0035L,
        +9.4974276355631964703077105664332465971 09e-0035L,
        -3.2831705231769986016150659533391526193 2e-0036L,
        -5.0172357093871904102901865304584289508 62e-0035L,
        -2.3914747976891091716228343016026401392 58e-0035L,
        -8.3505713576339088152988907379440838530 80e-0036L,
        +7.0367568890732650424217371906718764407 29e-0035L,
        -5.1824848530646464575368930185669561898 17e-0035L,
        +9.4222425486218320656921167363940648797 58e-0035L,
        -3.9675008253988623091673061302164182811 03e-0035L,
        +7.1435289915633006145232736150927672435 21e-0035L,
        +1.1598712528679851242465178341004443277 47e-0035L,
        +4.6969334783581154953097392132018744666 85e-0035L,
        -3.3865131759950047107992419849998191651 97e-0035L,
        -8.5873187742982470688686559351038744535 22e-0035L,
        -9.6059515487493505031854993622460690888 35e-0035L,
        +9.6097339321280127845075586971417858136 55e-0035L,
        +6.3783979214400284392447614497804858459 57e-0035L,
        +7.7924307856958642494564611251692777011 77e-0035L,
        +7.3613377675884565241319308366339321950 88e-0035L,
        -6.4729951479133472300352145756121705252 66e-0035L,
        +8.5874744179536986942787980622952296242 07e-0035L,
        +2.3718154228251748356916512283026909779 51e-0035L,
        -3.0268916820961187730045973734219003142 56e-0037L,
};
```

```
**********************************************************
   16912 Thu Jun  6 20:38:28 2019
new/usr/src/lib/libm/common/Q/_TBL_expm1l.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

/*
 * Table of constants for expm1l.
 * By K.C. Ng, June 30, 1995
 */

#include "libm.h"

const long double _TBL_expm1lx[] = {
        +7.8124999999999999999999999999995004619e-03L,
        +2.3437499999999999999999999999998790275e-02L,
        +3.9062499999999999999999999999981095794e-02L,
        +5.4687500000000000000000000000007637516e-02L,
        +7.0312500000000000000000000000001673152e-02L,
        +8.5937500000000000000000000000004285194e-02L,
        +1.0156249999999999999999999999997969348e-01L,
        +1.1718749999999999999999999999998630182e-01L,
        +1.3281249999999999999999999999999246480e-01L,
        +1.4843750000000000000000000000000096296e-01L,
        +1.6406249999999999999999999999999947037e-01L,
        +1.7968750000000000000000000000007190941e-01L,
        +1.9531249999999999999999999999999874815e-01L,
        +2.1093749999999999999999999999999877222e-01L,
        +2.2656250000000000000000000000000019259e-01L,
        +2.4218749999999999999999999999999800185e-01L,
        +2.5781249999999999999999999999996335918e-01L,
        +2.7343749999999999999999999999999903704e-01L,
        +2.8906249999999999999999999999998362960e-01L,
        +3.0468750000000000000000000000000240741e-01L,
        +3.2031249999999999999999999999999542592e-01L,
        +3.3593749999999999999999999999999619629e-01L,
```



```
        +3.5156250000000000000000000000001040002e-01L,
        +3.6718749999999999999999999999999316295e-01L,
        +3.8281250000000000000000000000000871483e-01L,
        +3.9843750000000000000000000000000385186e-01L,
        +4.1406249999999999999999999999999975926e-01L,
        +4.2968750000000000000000000000000385186e-01L,
        +4.4531250000000000000000000000000032119697e-01L,
        +4.6093749999999999999999999999999956667e-01L,
        +4.7656250000000000000000000000000134815e-01L,
        +4.9218749999999999999999999999999388517e-01L,
        +5.0781249999999999999999999999999778518e-01L,
        +5.2343749999999999999999999999999325925e-01L,
        +5.3906249999999999999999999999999990649610e-01L,
        +5.5468750000000000000000000000000452594e-01L,
        +5.7031250000000000000000000000000645187e-01L,
        +5.8593749999999999999999999999999807407e-01L,
        +6.0156250000000000000000000000000182963e-01L,
        +6.1718750000000000000000000000000048148e-01L,
        +6.3281250000000000000000000000000943706e-01L,
        +6.4843749999999999999999999999999470369e-01L,
        +6.6406250000000000000000000000000192593e-01L,
        +6.7968750000000000000000000000000693335e-01L,
        +6.9531250000000000000000000000001482966e-01L,
        +7.1093750000000000000000000000000154074e-01L,
        +7.2656250000000000000000000000000385186e-01L,
        +7.4218750000000000000000000000000115556e-01L,
        +7.5781250000000000000000000000000048148e-01L,
        +7.7343749999999999999999999999999942222e-01L,
        +7.8906249999999999999999999999999720740e-01L,
        +8.0468749999999999999999999999999634073e-01L,
        +8.2031249999999999999999999999999797777e-01L,
        +8.3593750000000000000000000000000828150e-01L,
        +8.5156249999999999999999999999999865185e-01L,
        +8.6718749999999999999999999999999682222e-01L,
        +8.8281249999999999999999999999999749629e-01L,
        +8.9843749999999999999999999999999836296e-01L,
        +9.1406249999999999999999999999999229628e-01L,
        +9.2968750000000000000000000000000462223e-01L,
        +9.4531249999999999999999999999999499258e-01L,
        +9.6093749999999999999999999999999894074e-01L,
        +9.7656249999999999999999999999999855555e-01L,
        +9.9218750000000000000000000000000028889e-01L,
        +1.0078124999999999999999999999999870963e+00L,
        +1.0234375000000000000000000000000003852e+00L,
        +1.0390624999999999999999999999999998074e+00L,
        +1.0546874999999999999999999999999801629e+00L,
        +1.0703125000000000000000000000000182963e+00L,
        +1.0859375000000000000000000000000021185e+00L,
        +1.1015624999999999999999999999999978815e+00L,
        +1.1171874999999999999999999999999986518e+00L,
        +1.1328124999999999999999999999999984593e+00L,
        +1.1484374999999999999999999999999980741e+00L,
        +1.1640625000000000000000000000000000000e+00L,
        +1.1796874999999999999999999999999998074e+00L,
        +1.1953125000000000000000000000000196445e+00L,
        +1.2109374999999999999999999999999976889e+00L,
        +1.2265625000000000000000000000000017333e+00L,
        +1.2421874999999999999999999999999976889e+00L,
        +1.2578124999999999999999999999999980741e+00L,
        +1.2734374999999999999999999999999951852e+00L,
        -7.8125000000000000000000000000074012886e-03L,
        -2.3437499999999999999999999999997418050e-02L,
        -3.9062499999999999999999999999998134255e-02L,
        -5.4687500000000000000000000000006361587e-02L,
        -7.0312500000000000000000000000006271309e-02L,
        -8.5937500000000000000000000000001721300e-02L,
```

```
        -1.015625000000000000000000000000020463e-01L,
        -1.171875000000000000000000000000290093e-01L,
        -1.328124999999999999999999999999987963e-01L,
        -1.484374999999999999999999999999942222e-01L,
        -1.640625000000000000000000000000026482e-01L,
        -1.796875000000000000000000000000052963e-01L,
        -1.953124999999999999999999999999867592e-01L,
        -2.109375000000000000000000000000081852e-01L,
        -2.265625000000000000000000000000250371e-01L,
        -2.421874999999999999999999999999718333e-01L,
        -2.578125000000000000000000000001059261e-01L,
        -2.734374999999999999999999999999557036e-01L,
        -2.890625000000000000000000000000520001e-01L,
        -3.046874999999999999999999999999792963e-01L,
        -3.203125000000000000000000000000231112e-01L,
        -3.359374999999999999999999999999383702e-01L,
        -3.515624999999999999999999999999903704e-01L,
        -3.671874999999999999999999999999634073e-01L,
        -3.828124999999999999999999999999956667e-01L,
        -3.984375000000000000000000000000004815e-01L,
        -4.140624999999999999999999999999870000e-01L,
        -4.296875000000000000000000000000216667e-01L,
        -4.453125000000000000000000000000072222e-01L,
        -4.609374999999999999999999999999513703e-01L,
        -4.765625000000000000000000000000539260e-01L,
        -4.921874999999999999999999999999759259e-01L,
        -5.078125000000000000000000000000067408e-01L,
        -5.234375000000000000000000000001386670e-01L,
        -5.390624999999999999999999999996427400e-01L,
        -5.546875000000000000000000000000404445e-01L,
        -5.703124999999999999999999999998536293e-01L,
        -5.859374999999999999999999999999634073e-01L,
        -6.015625000000000000000000000000028889e-01L,
        -6.171874999999999999999999999998468886e-01L,
        -6.328124999999999999999999999999335554e-01L,
        -6.484375000000000000000000000000356297e-01L,
        -6.640625000000000000000000000000019259e-01L,
        -6.796875000000000000000000000000067408e-01L,
        -6.953124999999999999999999999998439997e-01L,
        -7.109375000000000000000000000000009630e-01L,
        -7.265625000000000000000000000000414075e-01L,
        -7.421874999999999999999999999998738516e-01L,
        -7.578125000000000000000000000000019259e-01L,
        -7.734375000000000000000000000000808891e-01L,
        -7.890625000000000000000000000000028889e-01L,
        -8.046875000000000000000000000000096296e-01L,
        -8.203124999999999999999999999999634073e-01L,
        -8.359375000000000000000000000000086667e-01L,
        -8.515625000000000000000000000000115556e-01L,
        -8.671875000000000000000000000000028889e-01L,
        -8.828125000000000000000000000000577779e-01L,
        -8.984375000000000000000000000000587409e-01L,
        -9.140625000000000000000000000000654816e-01L,
        -9.296874999999999999999999999999210369e-01L,
        -9.453125000000000000000000000000115556e-01L,
        -9.609374999999999999999999999999682222e-01L,
        -9.765625000000000000000000000000298519e-01L,
        -9.921874999999999999999999999999711111e-01L,
        -1.007812499999999999999999999999953778e+00L,
        -1.023437500000000000000000000000279260e+00L,
        -1.039062500000000000000000000000000000e+00L,
        -1.054687500000000000000000000000042370e+00L,
        -1.070312499999999999999999999999969185e+00L,
        -1.085937499999999999999999999999965333e+00L,
        -1.101562500000000000000000000000144445e+00L,
        -1.117187500000000000000000000000007704e+00L,
```



```
        -1.132812500000000000000000000000052000e+00L,
        -1.148437500000000000000000000000021185e+00L,
        -1.164062500000000000000000000000188741e+00L,
        -1.179687499999999999999999999999915259e+00L,
        -1.195312500000000000000000000000196445e+00L,
        -1.210937499999999999999999999999965333e+00L,
        -1.226562499999999999999999999999940296e+00L,
        -1.242187499999999999999999999999955704e+00L,
        -1.257812500000000000000000000000036593e+00L,
        -1.273437500000000000000000000000005778e+00L,
};

#endif /* ! codereview */
const long double _TBL_expm1l[] = {
        +7.843097206447977693453559760123075745 5e-03L,
        +2.371431660235791696885053216576771968 4e-02L,
        +3.983547133623000057662200987517213529 5e-02L,
        +5.621049731693197118133670376813783996 9e-02L,
        +7.284339243487744441130009501081556201 4e-02L,
        +8.973821753809323101828202541851687407 3e-02L,
        +1.068990974236574827876023937417564864 2e-01L,
        +1.243302218447507174517329054449411524 1e-01L,
        +1.420358465335655896761614313487918936 0e-01L,
        +1.600202942403251470292152138940969271 3e-01L,
        +1.782879557886632426631433081958552851 1e-01L,
        +1.968432911476247711821968527701544105 8e-01L,
        +2.156908305205474518300182545403953108 2e-01L,
        +2.348351754510910046840161108760012261 4e-01L,
        +2.542809999466837520048212530906858730 1e-01L,
        +2.740330516196609292764965387163374691 5e-01L,
        +2.940961528463733098261106268100235482 2e-01L,
        +3.144752019445491342841349219798716929 0e-01L,
        +3.351751743691967926110409106635916383 5e-01L,
        +3.562011239273402330584882593857602661 8e-01L,
        +3.775581840118836703607622378892485508 3e-01L,
        +3.992515688549068357862059454951842577 7e-01L,
        +4.212865748006967555595399377075915671 6e-01L,
        +4.436685815988268627523668434427654229 1e-01L,
        +4.664030537175991422072580622261900134 3e-01L,
        +4.894955416781699796055764474013711118 0e-01L,
        +5.129516834096854356243837541143326897 0e-01L,
        +5.367772056257567954863095631977628167 7e-01L,
        +5.609779252226124543404296589534373438 5e-01L,
        +5.855597506992674910940477323437945428 2e-01L,
        +6.105286836000576588368576715139011721 0e-01L,
        +6.358908199798899801726944886054703041 9e-01L,
        +6.616523518925676819330374640390138734 6e-01L,
        +6.878195689025528372446796287495634487 5e-01L,
        +7.143988596205358051369170173111114417 2e-01L,
        +7.413967132631863701982922765711310993 8e-01L,
        +7.688197212374673886484036560813411439 8e-01L,
        +7.966745787498977440179624291990629311 9e-01L,
        +8.249680864411575068942484398874736973 3e-01L,
        +8.537071520464343803765299877394721783 3e-01L,
        +8.828987920819167975074551877605232271 4e-01L,
        +9.125501335578454205362498947689967848 6e-01L,
        +9.426684157185412759862975831367607278 2e-01L,
        +9.732609918098353457263977417630775976 2e-01L,
        +1.004335330874331124189699604163548448 6e+00L,
        +1.035899019574938447180307894277853984 6e+00L,
        +1.067959764047123800013898295403223914 9e+00L,
        +1.100525391780329323740530210984309522 8e+00L,
        +1.133603853529019859606908279647588637 2e+00L,
        +1.167203225253824605470282665302685605 8e+00L,
        +1.201331710093247305343031919769509885 9e+00L,
        +1.235997640366426356419592855632405185 3e+00L,
```

```
        +1.2712094796074337986378900692396989892e+00L,
        +1.3069758246316098519065681271390426992e+00L,
        +1.3433054076344374874670823571053750423e+00L,
        +1.3802070983234694643656048039370676504e+00L,
        +1.4176899060838283316233663173665695732e+00L,
        +1.4557629821778080933856445408266366139e+00L,
        +1.4944356219791145601003068865880283870e+00L,
        +1.5337172672422898656593276319684411587e+00L,
        +1.5736175084078752204957296405891018055e+00L,
        +1.6141460869438746959611789503655886752e+00L,
        +1.6553128977240916980462361132966809382e+00L,
        +1.6971279914439187908098398388318058200e+00L,
        +1.7396015770741706739548391625061335312e+00L,
        +1.7827440243535594070547844776995992896e+00L,
        +1.8265658663204204072713762882878497127e+00L,
        +1.8710778018843073303053045700866748054e+00L,
        +1.9162906984380836781353147778063807290e+00L,
        +1.9622155945111488641976761209920359164e+00L,
        +2.0088637024644465094546949911966296733e+00L,
        +2.0562464112279129437484158431835361592e+00L,
        +2.1043752890810342484246997824723919209e+00L,
        +2.1532620864771907009732038831535016010e+00L,
        +2.2029187389124781729299756967091584890e+00L,
        +2.2533573698397068911414662577557965622e+00L,
        +2.3045029362828900234289201885433322512e+00L,
        +2.3566300185707375845529318204056188573e+00L,
        +2.4094892499365111287660263409108200844e+00L,
        +2.4631808930739490736114262571853290077e+00L,
        +2.5177180565610557168667363422785819682e+00L,
        +2.5731140554059017538295927080989585090e+00L,
        -7.7820617397564878940627738863895136168e-03L,
        -2.3164975049937966141654020345517900132e-02L,
        -3.8309398394574704340244721980137502162e-02L,
        -5.3219029217871103345945692892173875140e-02L,
        -6.7897507640472422098597150880870814431e-02L,
        -8.2348417348184187852664478898998721220e-02L,
        -9.6575286466913289047103004903893595054e-02L,
        -1.1058158842404436382535801893754083366e-01L,
        -1.2437074279646178545389116639858446817e-01L,
        -1.3794611614542428546897208319214646321e-01L,
        -1.5131102283849604551092782942638950286e-01L,
        -1.6446872585873492869892658849405193342e-01L,
        -1.7742243760133541028616024893906020644e-01L,
        -1.9017532065792070445541830028432254804e-01L,
        -2.0273048858867556872072433107848955384e-01L,
        -2.1509100668250829875574108587424583890e-01L,
        -2.2725989270542750384925893490015094814e-01L,
        -2.3924011763731637587872084997993792064e-01L,
        -2.5103460639728433199192216502316397159e-01L,
        -2.6264623855777312240316411149867979990e-01L,
        -2.7407784904759174916236707216223561007e-01L,
        -2.8533222884405183877364930326430409924e-01L,
        -2.9641212565437245046372839566093886675e-01L,
        -3.0732024458652068208051596680746383072e-01L,
        -3.1805924880965185639883349668276001738e-01L,
        -3.2863176020431053139357768964876694788e-01L,
        -3.3904036000255107819096872094974754682e-01L,
        -3.4928758941813410931539692705947339063e-01L,
        -3.5937595026695261691197788694876862518e-01L,
        -3.6930790557783929525168292907013160482e-01L,
        -3.7908588019390417343445352368401927192e-01L,
        -3.8871226136454937222268435896035706784e-01L,
        -3.9818939932830552279841068785563696096e-01L,
        -4.0751960788663214438675145523141552015e-01L,
        -4.1670516496882207157854692957232201487e-01L,
        -4.2574831318814785027807613334156070311e-01L,
```



```
        -4.3465126038938588296698375694570623290e-01L,
        -4.4341618018785199889104786302733041801e-01L,
        -4.5204521250008005232823143909224611574e-01L,
        -4.6054046406627311177860364731167719334e-01L,
        -4.6890400896465479423105583940253757841e-01L,
        -4.7713788911784632111786130180348641941e-01L,
        -4.8524411479139292568322765140558720714e-01L,
        -4.9322466508456132479074344820321183415e-01L,
        -5.0108148841352808120907176750144850100e-01L,
        -5.0881650298707682468469397665961648404e-01L,
        -5.1643159727492047118140006480748942245e-01L,
        -5.2392863046876277909203259884139061144e-01L,
        -5.3130943293621180856911747373956708600e-01L,
        -5.3857580666765610494466923948125161529e-01L,
        -5.4572952571621270908956480845806990458e-01L,
        -5.5277233663085440607996537742519552762e-01L,
        -5.5970595888282195827936433920573211501e-01L,
        -5.6653208528542542950338319090135722295e-01L,
        -5.7325238240733709291016556235278458118e-01L,
        -5.7986849097947682625807457952262425338e-01L,
        -5.8638202629558933380643321697780798620e-01L,
        -5.9279457860661099402232802681583606392e-01L,
        -5.9910771350892261602101192647121362362e-01L,
        -6.0532297232658289493900937468812013977e-01L,
        -6.1144187248763588685320419660027392238e-01L,
        -6.1746590789458437705718363441235796711e-01L,
        -6.2339654928911959113469156706514092208e-01L,
        -6.2923524461119629598145525663487507890e-01L,
        -6.3498341935254095737853086332102468069e-01L,
        -6.4064247690467926157560858675744165996e-01L,
        -6.4621379890156797026959945044832806294e-01L,
        -6.5169874555691476103560779451668832748e-01L,
        -6.5709865599626840836245660190447925603e-01L,
        -6.6241484858396038364664323096343822726e-01L,
        -6.6764862124497769549523628863318702845e-01L,
        -6.7280125178184555417288842186006604551e-01L,
        -6.7787399818659722569870081614203949644e-01L,
        -6.8286809894790724165768340684449330930e-01L,
        -6.8778477335346294994617200009875163606e-01L,
        -6.9262522178764822913249147051072428209e-01L,
        -6.9739062602461204459956440664519817754e-01L,
        -7.0208214951679339786562882012396622127e-01L,
        -7.0670093767897311117765076057926938575e-01L,
        -7.1124811816792179736856679611852620191e-01L,
        -7.1572480115771228979753508374143873445e-01L,
        -7.2013207961076374868941367244540733250e-01L,
};
```

```
**********************************************************
   29212 Thu Jun  6 20:38:28 2019
new/usr/src/lib/libm/common/Q/_TBL_ipio2l.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

/*
 * Table of constants for 2/pi, used in __rem_pio2l (trigl) function.
 * By K.C. Ng, April 25, 1989
 */

#include "libm.h"

const int _TBL_ipio2l_inf[] = {         /* by DHBailey MP package */
        0xA2F983, 0x6E4E44, 0x1529FC, 0x2757D1, 0xF534DD, 0xC0DB62,
        0x95993C, 0x439041, 0xFE5163, 0xABDEBB, 0xC561B7, 0x246E3A,
        0x424DD2, 0xE00649, 0x2EEA09, 0xD1921C, 0xFE1DEB, 0x1CB129,
        0xA73EE8, 0x8235F5, 0x2EBB44, 0x84E99C, 0x7026B4, 0x5F7E41,
        0x3991D6, 0x398353, 0x39F49C, 0x845F8B, 0xBDF928, 0x3B1FF8,
        0x97FFDE, 0x05980F, 0xEF2F11, 0x8B5A0A, 0x6D1F6D, 0x367ECF,
        0x27CB09, 0xB74F46, 0x3F669E, 0x5FEA2D, 0x7527BA, 0xC7EBE5,
        0xF17B3D, 0x0739F7, 0x8A5292, 0xEA6BFB, 0x5FB11F, 0x8D5D08,
        0x560330, 0x46FC7B, 0x6BABF0, 0xCFBC20, 0x9AF436, 0x1DA9E3,
        0x91615E, 0xE61B08, 0x659985, 0x5F14A0, 0x68408D, 0xFFD880,
        0x4D7327, 0x310606, 0x1556CA, 0x73A8C9, 0x60E27B, 0xC08C6B,
        0x47C419, 0xC367CD, 0xDCE809, 0x2A8359, 0xC4768B, 0x961CA6,
        0xDDAF44, 0xD15719, 0x053EA5, 0xFF0705, 0x3F7E33, 0xE832C2,
        0xDE4F98, 0x327DBB, 0xC33D26, 0xEF6B1E, 0x5EF89F, 0x3A1F35,
        0xCAF27F, 0x1D87F1, 0x21907C, 0x7C246A, 0xFA6ED5, 0x772D30,
        0x433B15, 0xC614B5, 0x9D19C3, 0xC2C4AD, 0x414D2C, 0x5D000C,
        0x467D86, 0x2D71E3, 0x9AC69B, 0x006233, 0x7CD2B4, 0x97A7B4,
        0xD55537, 0xF63ED7, 0x1810A3, 0xFC764D, 0x2A9D64, 0xABD770,
        0xF87C63, 0x57B07A, 0xE71517, 0x5649C0, 0xD9D63B, 0x3884A7,
        0xCB2324, 0x778AD6, 0x23545A, 0xB91F00, 0x1B0AF1, 0xDFCE19,
        0xFF319F, 0x6A1E66, 0x615799, 0x47FBAC, 0xD87F7E, 0xB76522,
        0x89E832, 0x60BFE6, 0xCDC4EF, 0x09366C, 0xD43F5D, 0xD7DE16,
```



```
        0xDE3B58, 0x929BDE, 0x2822D2, 0xE88628, 0x4D58E2, 0x32CAC6,
        0x16E308, 0xCB7DE0, 0x50C017, 0xA71DF3, 0x5BE018, 0x34132E,
        0x621283, 0x014883, 0x5B8EF5, 0x7FB0AD, 0xF2E91E, 0x434A48,
        0xD36710, 0xD8DDAA, 0x425FAE, 0xCE616A, 0xA4280A, 0xB499D3,
        0xF2A606, 0x7F775C, 0x83C2A3, 0x883C61, 0x78738A, 0x5A8CAF,
        0xBDD76F, 0x63A62D, 0xCBBFF4, 0xEF818D, 0x67C126, 0x45CA55,
        0x36D9CA, 0xD2A828, 0x8D61C2, 0x77C912, 0x142604, 0x9B4612,
        0xC459C4, 0x44C5C8, 0x91B24D, 0xF31700, 0xAD43D4, 0xE54929,
        0x10D5FD, 0xFCBE00, 0xCC941E, 0xEECE70, 0xF53E13, 0x80F1EC,
        0xC3E7B3, 0x28F8C7, 0x940593, 0x3E71C1, 0xB3092E, 0xF3450B,
        0x9C1288, 0x7B20AB, 0x9FB52E, 0xC29247, 0x2F327B, 0x6D550C,
        0x90A772, 0x1FE76B, 0x96CB31, 0x4A1679, 0xE27941, 0x89DFF4,
        0x9794E8, 0x84E6E2, 0x973199, 0x6BED88, 0x365F5F, 0x0EFDBB,
        0xB49A48, 0x6CA467, 0x427271, 0x325D8D, 0xB8159F, 0x09E5BC,
        0x25318D, 0x3974F7, 0x1C0530, 0x010C0D, 0x68084B, 0x58EE2C,
        0x90AA47, 0x02E774, 0x24D6BD, 0xA67DF7, 0x72486E, 0xEF169F,
        0xA6948E, 0xF691B4, 0x5153D1, 0xF20ACF, 0x339820, 0x7E4BF5,
        0x6863B2, 0x5F3EDD, 0x035D40, 0x7F8985, 0x295255, 0xC06437,
        0x10D86D, 0x324832, 0x754C5B, 0xD4714E, 0x6E5445, 0xC1090B,
        0x69F52A, 0xD56614, 0x9D0727, 0x50045D, 0xDB3BB4, 0xC576EA,
        0x17F987, 0x7D6B49, 0xBA271D, 0x296996, 0xACCCC6, 0x5414AD,
        0x6AE290, 0x89D988, 0x50722C, 0xBEA404, 0x940777, 0x7030F3,
        0x27FC00, 0xA871EA, 0x49C266, 0x3DE064, 0x83DD97, 0x973FA3,
        0xFD9443, 0x8C860D, 0xDE4131, 0x9D3992, 0x8C70DD, 0xE7B717,
        0x3BDF08, 0x2B3715, 0xA0805C, 0x93805A, 0x921110, 0xD8E80F,
        0xAF806C, 0x4BFFDB, 0x0F9038, 0x761859, 0x15A562, 0xBBCB61,
        0xB989C7, 0xBD4010, 0x04F2D2, 0x277549, 0xF6B6EB, 0xBB22DB,
        0xAA140A, 0x2F2689, 0x768364, 0x333B09, 0x1A940E, 0xAA3A51,
        0xC2A31D, 0xAEEDAF, 0x12265C, 0x4DC26D, 0x9C7A2D, 0x9756C0,
        0x833F03, 0xF6F009, 0x8C402B, 0x99316D, 0x07B439, 0x15200C,
        0x5BC3D8, 0xC492F5, 0x4BADC6, 0xA5CA4E, 0xCD37A7, 0x36A9E6,
        0x9492AB, 0x6842DD, 0xDE6319, 0xEF8C76, 0x528B68, 0x37DBFC,
        0xABA1AE, 0x3115DF, 0xA1AE00, 0xDAFB0C, 0x664D64, 0xB705ED,
        0x306529, 0xBF5657, 0x3AFF47, 0xB9F96A, 0xF3BE75, 0xDF9328,
        0x3080AB, 0xF68C66, 0x15CB04, 0x0622FA, 0x1DE4D9, 0xA4B33D,
        0x8F1B57, 0x09CD36, 0xE9424E, 0xA4BE13, 0xB52333, 0x1AAAF0,
        0xA8654F, 0xA5C1D2, 0x0F3F0B, 0xCD785B, 0x76F923, 0x048B7B,
        0x721789, 0x53A6C6, 0xE26E6F, 0x00EBEF, 0x584A9B, 0xB7DAC4,
        0xBA66AA, 0xCFCF76, 0x1D02D1, 0x2DF1B1, 0xC1998C, 0x77ADC3,
        0xDA4886, 0xA05DF7, 0xF480C6, 0x2FF0AC, 0x9AECDD, 0xBC5C3F,
        0x6DDED0, 0x1FC790, 0xB6DB2A, 0x3A25A3, 0x9AAF00, 0x9353AD,
        0x0457B6, 0xB42D29, 0x7E804B, 0xA707DA, 0x0EAA76, 0xA1597B,
        0x2A1216, 0x2DB7DC, 0xFDE5FA, 0xFEDB89, 0xFDBE89, 0x6C76E4,
        0xFCA906, 0x70803E, 0x156E85, 0xFF87FD, 0x073E28, 0x336761,
        0x86182A, 0xEABD4D, 0xAFE7B3, 0x6E6D8F, 0x396795, 0x5BBF31,
        0x48D784, 0x16DF30, 0x432DC7, 0x356125, 0xCE70C9, 0xB8CB30,
        0xFD6CBF, 0xA200A4, 0xE46C05, 0xA0DD5A, 0x476F21, 0xD21262,
        0x845CB9, 0x496170, 0xE0566B, 0x015299, 0x375550, 0xB7D51E,
        0xC4F133, 0x5F6E13, 0xE4305D, 0xA92E85, 0xC3B21D, 0x3632A1,
        0xA4B708, 0xD4B1EA, 0x21F716, 0xE4698F, 0x77FF27, 0x80030C,
        0x2D408D, 0xA0CD4F, 0x99A520, 0xD3A2B3, 0x0A5D2F, 0x42F9B4,
        0xCBDA11, 0xD0BE7D, 0xC1DB9B, 0xBD17AB, 0x81A2CA, 0x5C6A08,
        0x17552E, 0x550027, 0xF0147F, 0x8607E1, 0x640B14, 0x8D4196,
        0xDEBE87, 0x2AFDDA, 0xB6256B, 0x34897B, 0xFEF305, 0x9EBFB9,
        0x4F6A68, 0xA82A4A, 0x5AC44F, 0xBCF82D, 0x985AD7, 0x95C7F4,
        0x8D4D0D, 0xA63A20, 0x5F57A4, 0xB13F14, 0x953880, 0x0120CC,
        0x86DD71, 0xB6DEC9, 0xF560BF, 0x11654D, 0x6B0701, 0xACB08C,
        0xD0C0B2, 0x485551, 0x0EFB1E, 0xC37295, 0x3B06A3, 0x3540C0,
        0x7BDC06, 0xCC45E0, 0xFA294E, 0xC8CAD6, 0x41F3E8, 0xDE647C,
        0xD8649B, 0x31BED9, 0xC397A4, 0xD45877, 0xC5E369, 0x13DAF0,
        0x3C3ABA, 0x461846, 0x5F7555, 0xF5BDD2, 0xC6926E, 0x5D2EAC,
        0xED440E, 0x423E1C, 0x87C461, 0xE9FD29, 0xF3D6E7, 0xCA7C22,
        0x35916F, 0xC5E008, 0x8DD7FF, 0xE26A6E, 0xC6FDB0, 0xC10893,
        0x745D7C, 0xB2AD6B, 0x9D6ECD, 0x7B723E, 0x6A11C6, 0xA9CFF7,
        0xDF7329, 0xBAC9B5, 0x5100B7, 0x0DB2E2, 0x24BA74, 0x607DE5,
        0x8AD874, 0x2C150D, 0x0C1881, 0x94667E, 0x162901, 0x767A9F,
```

```
        0xBEFDFD, 0xEF4556, 0x367ED9, 0x13D9EC, 0xB9BA8B, 0xFC97C4,
        0x27A831, 0xC36EF1, 0x36C594, 0x56A8D8, 0xB5A8B4, 0x0ECCCF,
        0x2D8912, 0x34576F, 0x89562C, 0xE3CE99, 0xB920D6, 0xAA5E6B,
        0x9C2A3E, 0xCC5F11, 0x4A0BFD, 0xFBF4E1, 0x6D3B8E, 0x2C86E2,
        0x84D4E9, 0xA9B4FC, 0xD1EEEF, 0xC9352E, 0x61392F, 0x442138,
        0xC8D91B, 0x0AFC81, 0x6A4AFB, 0xD81C2F, 0x84B453, 0x8C994E,
        0xCC2254, 0xDC552A, 0xD6C6C0, 0x96190B, 0xB8701A, 0x649569,
        0x605A26, 0xEE523F, 0x0F117F, 0x11B5F4, 0xF5CBFC, 0x2DBC34,
        0xEEBC34, 0xCC5DE8, 0x605EDD, 0x9B8E67, 0xEF3392, 0xB817C9,
        0x9B5861, 0xBC57E1, 0xC68351, 0x103ED8, 0x4871DD, 0xDD1C2D,
        0xA118AF, 0x462C21, 0xD7F359, 0x987AD9, 0xC0549E, 0xFA864F,
        0xFC0656, 0xAE79E5, 0x362289, 0x22AD38, 0xDC9367, 0xAAE855,
        0x382682, 0x9BE7CA, 0xA40D51, 0xB13399, 0x0ED7A9, 0x480569,
        0xF0B265, 0xA7887F, 0x974C88, 0x36D1F9, 0xB39221, 0x4A827B,
        0x21CF98, 0xDC9F40, 0x5547DC, 0x3A74E1, 0x42EB67, 0xDF9DFE,
        0x5FD45E, 0xA4677B, 0x7AACBA, 0xA2F655, 0x23882B, 0x55BA41,
        0x086E59, 0x862A21, 0x834739, 0xE6E389, 0xD49EE5, 0x40FB49,
        0xE956FF, 0xCA0F1C, 0x8A59C5, 0x2BFA94, 0xC5C1D3, 0xCFC50F,
        0xAE5ADB, 0x86C547, 0x624385, 0x3B8621, 0x94792C, 0x876110,
        0x7B4C2A, 0x1A2C80, 0x12BF43, 0x902688, 0x893C78, 0xE4C4A8,
        0x7BDBE5, 0xC23AC4, 0xEAF426, 0x8A67F7, 0xBF920D, 0x2BA365,
        0xB1933D, 0x0B7CBD, 0xDC51A4, 0x63DD27, 0xDDE169, 0x19949A,
        0x9529A8, 0x28CE68, 0xB4ED09, 0x209F44, 0xCA984E, 0x638270,
        0x237C7E, 0x32B90F, 0x8EF5A7, 0xE75614, 0x08F121, 0x2A9DB5,
        0x4D7E6F, 0x5119A5, 0xABF9B5, 0xD6DF82, 0x61DD96, 0x023616,
        0x9F3AC4, 0xA1A283, 0x6DED72, 0x7A8D39, 0xA9B882, 0x5C326B,
        0x5B2746, 0xED3400, 0x7700D2, 0x55F4FC, 0x4D5901, 0x8071E0,
        0xE13F89, 0xB295F3, 0x64A8F1, 0xAEA74B, 0x38FC4C, 0xEAB2BB,
        0x47270B, 0xABC3A7, 0x34BA60, 0x52DD34, 0xF8563A, 0xEB7E8A,
        0x31BB36, 0x5895B7, 0x47F7A9, 0x94C3AA, 0xD39225, 0x1E7F3E,
        0xD8974E, 0xBBA94F, 0xD8AE01, 0xE661B4, 0x393D8E, 0xA523AA,
        0x33068E, 0x1633B5, 0x3BB188, 0x1D3A9D, 0x4013D0, 0xCC1BE5,
        0xF862E7, 0x3BF28F, 0x39B5BF, 0x0BC235, 0x22747E, 0xA247C0,
        0xD52D1F, 0x19ADD3, 0x9094DF, 0x9311D0, 0xB42B25, 0x496DB2,
        0xE264B2, 0x5EF135, 0x3BC6A4, 0x1A4AD0, 0xAAC92E, 0x64E886,
        0x573091, 0x982CFB, 0x311B1A, 0x08728B, 0xBDCEE1, 0x60E142,
        0xEB641D, 0xD0BBA3, 0xE559D4, 0x597B8C, 0x2A4483, 0xF332BA,
        0xF84867, 0x2C8D1B, 0x2FA9B0, 0x50F3DD, 0xF9F573, 0xDB61B4,
        0xFE233E, 0x6C41A6, 0xEEA318, 0x775A26, 0xBC5E5C, 0xCEA708,
        0x94DC57, 0xE20196, 0xF1E839, 0xBE4851, 0x5D2D2F, 0x4E9555,
        0xD96EC2, 0xE7D755, 0x6304E0, 0xC02E0E, 0xFC40A0, 0xBBF9B3,
        0x7125A7, 0x222DFB, 0xF619D8, 0x838C1C, 0x6619E6, 0xB20D55,
        0xBB5137, 0x79E809, 0xAF9149, 0x0D73DE, 0x0B0DA5, 0xCE7F58,
        0xAC1934, 0x724667, 0x7A1A13, 0x9E26BC, 0x4555E7, 0x585CB5,
        0x711D14, 0x486991, 0x480D60, 0x56ADAB, 0xD62F64, 0x96EE0C,
        0x212FF3, 0x5D6D88, 0xA67684, 0x95651E, 0xAB9E0A, 0x4DDEFE,
        0x571010, 0x836A39, 0xF8EA31, 0x9E381D, 0xEAC8B1, 0xCAC96B,
        0x37F21E, 0xD505E9, 0x984743, 0x9FC56C, 0x0331B7, 0x3B8BF8,
        0x86E56A, 0x8DC343, 0x6230E7, 0x93CFD5, 0x6A8F2D, 0x733005,
        0x1AF021, 0xA09FCB, 0x7415A1, 0xD56B23, 0x6FF725, 0x2F4BC7,
        0xB8A591, 0x7FAC59, 0x5C55DE, 0x212C38, 0xB13296, 0x5CFF50,
        0x366262, 0xFA7B16, 0xF4D9A6, 0x2ACFE7, 0xF07403, 0xD4D604,
        0x6FD916, 0x31B1BF, 0xCBB450, 0x5BD7C8, 0x0CE194, 0x6BD643,
        0x4FD91C, 0xDF4543, 0x5F3453, 0xE2B5AA, 0xC9AEC8, 0x131485,
        0xF9D2BF, 0xBADB9E, 0x76F5B9, 0xAF15CF, 0xCA3182, 0x14B56D,
        0xE9FE4D, 0x50FC35, 0xF5AED5, 0xA2D0C1, 0xC96057, 0x192EB6,
        0xE91D92, 0x07D144, 0xAEA3C6, 0x343566, 0x26D5B4, 0x3161E2,
        0x37F1A2, 0x209EFF, 0x958E23, 0x493798, 0x35F4A6, 0x4BDC02,
        0xC2BE13, 0xBE80A0, 0x0B72A3, 0x115C5F, 0x1E1BD1, 0x0DB4D3,
        0x869E85, 0x96976B, 0x2AC91F, 0x8A26C2, 0x3070F0, 0x041412,
        0xFC9FA5, 0xF72A38, 0x9C6878, 0xE2AA76, 0x50CFE1, 0x559274,
        0x934E38, 0x0A92F7, 0x5533F0, 0xA63DB4, 0x399971, 0xE2B755,
        0xA98A7C, 0x008F19, 0xAC54D2, 0x2EA0B4, 0xF5F3E0, 0x60C849,
        0xFFD269, 0xAE52CE, 0x7A5FDD, 0xE9CE06, 0xFB0AE8, 0xA50CCE,
        0xEA9D3E, 0x3766DD, 0xB834F5, 0x0DA090,
};
```



```
#if 0
const int _TBL_ipio2l_66[] = {
        0xA2F983, 0x6E4E44, 0x152A00, 0x062BC4, 0x0DA276, 0xBED4C1,
        0xFDF905, 0x5CD5BA, 0x767CEC, 0x1F80D6, 0xC26053, 0x3A0070,
        0x107C2A, 0xF68EE9, 0x687B7A, 0xB990AA, 0x38DE4B, 0x96CFF3,
        0x92735E, 0x8B34F6, 0x195BFC, 0x27F88E, 0xA93EC5, 0x3958A5,
        0x3E5D13, 0x1C55A8, 0x5B4A8B, 0xA42E04, 0x12D105, 0x35580D,
        0xF62347, 0x450900, 0xB98BCA, 0xF7E8A4, 0xA2E5D5, 0x69BC52,
        0xF0381D, 0x1A0A88, 0xFE8714, 0x7F6735, 0xBB7D4D, 0xC6F642,
        0xB27E80, 0x6191BF, 0xB6B750, 0x52776E, 0xD60FD0, 0x607DCC,
        0x68BFAF, 0xED69FC, 0x6EB305, 0xD2557D, 0x25BDFB, 0x3E4AA1,
        0x84472D, 0x8B0376, 0xF77740, 0xD290DF, 0x15EC8C, 0x45A5C3,
        0x6181EF, 0xC5E7E8, 0xD8909C, 0xF62144, 0x298428, 0x6E5D9D,
        0xF9A9B4, 0xCDBD2F, 0xC083E7, 0x0D3957, 0xECA3B2, 0x96223C,
        0xC1080D, 0x087D47, 0x7D7576, 0xA614B1, 0x42A4B6, 0xAA173C,
        0xE217E5, 0xFDCD34, 0x279D5F, 0x39AACA, 0x1CA8DF, 0x8B6633,
        0x5C49E4, 0xB56803, 0x1E7938, 0x741FDC, 0x4CB19B, 0xCECC3B,
        0x921EB7, 0x7C0FC3, 0x361F23, 0xF9EE22, 0xBA4235, 0xA5FCA3,
        0xBD4680, 0xFCDF65, 0xFC96AD, 0x31C90C, 0x919EEB, 0xFE0FB7,
        0x75B4B0, 0x693961, 0x75BCAA, 0xEB6F39, 0xA343C0, 0xD16FF2,
        0x33DAD0, 0xC1E095, 0x053182, 0x11E4A1, 0x40F943, 0x32D314,
        0xAF1B98, 0xE1B05A, 0xE5F3AD, 0x6E633F, 0x363D14, 0xA3777C,
        0xC8C6EE, 0x001E18, 0x0D180C, 0xAA1369, 0xEDFBA2, 0x998A9D,
        0x16E799, 0x693B75, 0x90EF50, 0x938DD4, 0xFB7ACD, 0x67CEEB,
        0x249DE3, 0x9B9B52, 0xD8CDAC, 0xC31A54, 0x855FBF, 0x848591,
        0x0954B0, 0x946B8C, 0xA4C7B4, 0x9A9E51, 0xF20425, 0xAA2637,
        0xFC6657, 0x7D8625, 0x620B74, 0x8B578D, 0xEC9A05, 0xDEF24F,
        0x7F19B0, 0xFC2544, 0x1DA0F1, 0x23790C, 0xC4294D, 0x6D3C32,
        0x66FE56, 0xD45562, 0x66264F, 0xA24162, 0x13E930, 0xB0E7C0,
        0xFA0E97, 0xBFC62C, 0x0E663F, 0x90F33B, 0x55E73C, 0xD791F7,
        0xD3F00D, 0xAB01C7, 0x40CF8F, 0xA593BA, 0xE627D5, 0x4A8308,
        0x32DC06, 0x80C876, 0x1C3DB5, 0xB5489F, 0x632CDF, 0xB02517,
        0xD17EFA, 0x92570F, 0xFAED44, 0x8F8536, 0x27069B, 0xC014DC,
        0x997D48, 0x961D61, 0x7A960B, 0x31B622, 0xD3C425, 0xA69520,
        0x98D29E, 0xF1C973, 0x5483D7, 0x99611E, 0xEAFF5F, 0x7DEFF1,
        0x98475C, 0x91C787, 0x537E17, 0x068C65, 0xF05E52, 0x942F04,
        0x37CF92, 0xEF4223, 0xC4C52F, 0x521DAA, 0xBAAF97, 0x972236,
        0xA2B3D3, 0x62C921, 0x8D3A8B, 0x2B3302, 0x6061B9, 0x0CBE94,
        0x75F451, 0xBD06DE, 0x86042D, 0xFB61ED, 0x4C8869, 0x590232,
        0x479963, 0x23518D, 0xAF5D28, 0x60C9DE, 0x473DB0, 0x9DE009,
        0xD8FC4C, 0xE96991, 0x9CA455, 0x800BC8, 0x977CE0, 0xDCBFA6,
        0x19D249, 0xA0F76D, 0x5F9B2F, 0x452BB3, 0x77E091, 0xB6383A,
        0x7BE9C2, 0x4BF7C1, 0x8A5EBF, 0xEB0D55, 0x9AF4DC, 0x275CA0,
        0xED09D0, 0xE50A7F, 0xBEF42C, 0x4803AF, 0x56139F, 0xD58848,
        0x797D96, 0xB8352E, 0x49D90D, 0x7607E0, 0xC99256, 0x75F530,
        0xB72237, 0x1AF080, 0xC2E813, 0x06CFA9, 0xB9DF8E, 0x919C38,
        0x89D97E, 0x0464D5, 0xB12EEF, 0xD14165, 0x365A72, 0x550D35,
        0x3772D8, 0xF41B58, 0x0378A7, 0x2D5D7D, 0xD6E433, 0xDD2018,
        0x139FD7, 0x1B5621, 0x94E046, 0x97A323, 0x693176, 0x28DF59,
        0xD24273, 0x0E4E26, 0xA9A8F6, 0xF15B41, 0x450EE3, 0x57EA61,
        0x7DADA6, 0xF21086, 0x394BEE, 0x8F4813, 0x3FDEE9, 0xF3A53D,
        0xAB2F40, 0x8B1E2B, 0xA07FD4, 0x992CC4, 0x63532D, 0x9F35A2,
        0x6FA290, 0x0094DE, 0xD2A24D, 0x755B81, 0x79F9E1, 0xFE1D35,
        0xFEE8CC, 0x9224C5, 0x54E2CE, 0x41F31C, 0xF45138, 0xED6D10,
        0x6B439D, 0xD2BE46, 0xC327D4, 0x68BFB0, 0x46D5A5, 0x79B285,
        0x776D7C, 0xE18647, 0x00E32F, 0xEBB7F2, 0x5DE307, 0x5A8EA0,
        0x06CEFE, 0x20923C, 0x354CE1, 0xAD09C5, 0x56996D, 0xCFB124,
        0xEF7BC1, 0x76BF72, 0xF20753, 0x5BBAFA, 0xB8A2B2, 0x5914F2,
        0x5D834F, 0xE64A08, 0x14C3AB, 0x07796B, 0xF2212D, 0xC74049,
        0xB61C6A, 0x282CFC, 0x25070C, 0x315BF1, 0x6FEAD3, 0x2CD2E5,
        0xD10F9C, 0x1972BB, 0x908073, 0x0F368C, 0x69BE97, 0xA242B0,
        0x722DFE, 0xAFE6A2, 0x143D8B, 0x5C5699, 0x48232B, 0xFF49AC,
        0xB5FA62, 0x6AD778, 0x7A844D, 0x258AA0, 0x8EDE3D, 0x9A9496,
        0x49924E, 0xA33E97, 0x4F43FA, 0xC40741, 0x2F764A, 0x8EB2B1,
        0x8E67D3, 0x9FF324, 0x51B11B, 0x5D6E09, 0xE9AD3E, 0xFFA902,
```

```
        0xF48653, 0x0845D3, 0xDED33E, 0x32D30E, 0x6247CA, 0x7C586D,
        0x2EAF9E, 0x323A35, 0xAD11FB, 0x0F420C, 0x0E0685, 0x401B60,
        0xBB3D43, 0xF4D489, 0xBCDC4C, 0x40FFBA, 0x18AB08, 0x7AC72D,
        0x5E76DB, 0xE8344E, 0x3975A2, 0xF9611B, 0x1121F3, 0x3A429C,
        0x9B18EC, 0xF298B1, 0x8AEC78, 0x1C248B, 0x69108F, 0xDB2D37,
        0xA1A613, 0x910359, 0x521451, 0xD4441F, 0x0BB3B6, 0x50D9DB,
        0xBD589F, 0x62A62E, 0xA9B903, 0x935F63, 0x058BEC, 0x78BCB5,
        0x2CB460, 0x3A9037, 0x0291C4, 0x1FABC1, 0xBE7D05, 0xF948E7,
        0x6BA5CD, 0xF62A0A, 0x9AEA19, 0x2257AB, 0x2E0D7D, 0x9EB93F,
        0x5E3F77, 0xD4A13F, 0x08E3DB, 0xDFD689, 0x2B9B4E, 0xB58427,
        0x25424B, 0x1197FD, 0xCF298A, 0x314008, 0xD5687F, 0x0F0EAC,
        0x13C485, 0xF684B2, 0xED7EC7, 0x6E636D, 0x28C933, 0xE19058,
        0x688B6A, 0xC88905, 0xFB2F31, 0x61304C, 0xC19765, 0x60D81A,
        0x57F276, 0xC6EFC4, 0x048954, 0x303470, 0xDA6F6F, 0x93901A,
        0x911439, 0x363D12, 0x59E72B, 0x6F9F1E, 0x57C584, 0xDF0D23,
        0xBB743F, 0xADE99C, 0x546097, 0xFCC820, 0xCBB968, 0xDA9B5F,
        0x0DC271, 0x563337, 0x9ED662, 0xE7C44F, 0x3129F8, 0xF5EAF9,
        0xDAF7F2, 0xCD09FF, 0xA92535, 0x441C29, 0x7DF436, 0xE2B00A,
        0x36746F, 0xF1DC61, 0x9D3C9C, 0x63AB71, 0xB8F3BB, 0x1C80F6,
        0x62FF65, 0x5FFE5F, 0x3B2814, 0xBADE27, 0x1B384B, 0x268AA9,
        0xBD91EF, 0xCA436B, 0xABE107, 0x88DCA6, 0xC3AFC0, 0x85D155,
        0x464A48, 0xBFDAEB, 0xC6F389, 0x907C11, 0x0D3E41, 0xCD2197,
        0x549008, 0x817E4E, 0x8C7154, 0x1DC37F, 0x5E897E, 0xA9A2FE,
        0xEC6060, 0xCC0728, 0x430D3B, 0x62471C, 0xD3A4D3, 0x2BA57B,
        0xE5D15A, 0xD632F3, 0xF2B76F, 0xEC8498, 0xAE41C2, 0xAAF413,
        0xEAF5C0, 0xDD1B07, 0xB9A2A0, 0x59F230, 0xA3F61B, 0x8F8643,
        0x05DE6B, 0x1B5B8E, 0x63ECC5, 0xBFF01D, 0x8F1440, 0x3F8ADF,
        0x2E6539, 0xF3DB7A, 0x293FE5, 0x7EE714, 0x88E6D8, 0x2B2A6A,
        0xDF6E34, 0x8D4604, 0x4F6594, 0x639063, 0x6B51CC, 0x0D05CD,
        0x009607, 0xE7BF70, 0xC9A0EA, 0x0D80DD, 0xA1A065, 0x0DCB8F,
        0xA48430, 0x715934, 0x6FC8E4, 0x6FFC52, 0xEF8B05, 0xDE506A,
        0xE62BBC, 0x31480F, 0xEA64EA, 0x51E6FB, 0x9AE773, 0x21C54D,
        0xBFA080, 0x273D1E, 0x9FFD4E, 0x0C2CA8, 0x0690A5, 0xF8773B,
        0x4B2680, 0x6E3F56, 0xC8B89F, 0x0B7BD0, 0x71C8BF, 0x5AABD3,
        0x2BA93E, 0x9D2EE1, 0xCDF2FA, 0xEE57BE, 0x84A116, 0xDA756D,
        0x8FD6C0, 0x927153, 0xFF5EF3, 0x9F8331, 0x713411, 0xF945F3,
        0x0382B2, 0x8BAE30, 0xBC45A4, 0x630101, 0x5C9C3A, 0x643CFD,
        0x48115C, 0x17F03E, 0xB5F55E, 0x288DAF, 0x725660, 0xFB58E0,
        0xFC189E, 0x1ECA69, 0xFB19A6, 0xFA7A92, 0x7CC48E, 0x869372,
        0x58089A, 0x16DB5C, 0xADC0CD, 0x09D3D4, 0xD1108E, 0xDC64ED,
        0x3A999C, 0xAA8716, 0x5A3D8E, 0x7037FB, 0x1976AD, 0xE477D7,
        0x23782B, 0xC51F39, 0x4A5E9A, 0xDAD9DA, 0xE5B559, 0x08EF06,
        0x76E24F, 0x7361AD, 0x5F42A3, 0x9B70E5, 0xCE96C4, 0x552E99,
        0x6D7A6F, 0x804474, 0x4FA45B, 0x1D115B, 0x6D109E, 0x0A1A63,
        0x1084A6, 0xE18E5D, 0x2D8589, 0x203345, 0x4851AF, 0xA71EDC,
        0x03B6B1, 0x267970, 0xDEC908, 0x795BED, 0x7099B9, 0x209321,
        0x7FC2E7, 0x0F3E5E, 0xC7A4F4, 0x088129, 0x59AE63, 0x4E3251,
        0x344268, 0x79285D, 0x2B9494, 0xF1E2A2, 0xF7DA20, 0xDF6756,
        0xCA3BA3, 0x422489, 0xA2239C, 0x38724D, 0x2AC767, 0x601E9D,
        0xB47C6C, 0xA22481, 0xBBB655, 0x1EC0C4, 0xD84A97, 0xD449EE,
        0x162C9D, 0x782F29, 0xCEB4FA, 0xE317BC, 0x2FFDBD, 0xB342D2,
        0xB2CB19, 0x323AB9, 0x1AFF93, 0x13A8DF, 0x86B5A5, 0x5741D6,
        0xC54342, 0x3CAC29, 0xF7517C, 0x129A7A, 0xB2B8B4, 0x9B709F,
        0x3923C5, 0xEAFA6E, 0xDB9077, 0x29EEA0, 0x702D8C, 0x4DC14F,
        0xE46933, 0xA764E4, 0x754266, 0xFA4F98, 0x643DA5, 0xCA775C,
        0x7F1632, 0xE671A3, 0x4BF4C6, 0xA82378, 0xEFD317, 0xE62D38,
        0xD461C9, 0x8EEC80, 0xC89882, 0x4CC73C, 0x830F3F, 0xE4B200,
        0x582615, 0x6CD558, 0xA66727, 0xEF7975, 0xFEA5CE, 0x147A40,
        0x4796E4, 0xC07761, 0xF5D5B3, 0x6B65FB, 0xE4F14D, 0xA837CA,
        0x9A152A, 0x554E94, 0x83EC5F, 0xA62174, 0x85E2ED, 0xCCE71C,
        0x3540FF, 0x088A84, 0xBA2816, 0x293610, 0x4C3EE7, 0x8E55A9,
        0x49E5E5, 0x782178, 0x45D2AA, 0x9BB449, 0x00D282, 0xF61E67,
        0xE2F7DE, 0xCC6AA1, 0xCD1979, 0x52FEDB, 0x9A8776, 0x70A018,
        0x500271, 0x1273BA, 0xDE648E, 0x7AC7F7, 0x767725, 0xD0A457,
        0xF17250, 0xBC578C, 0x2DFD3A, 0x97F988, 0xA576C8, 0x8129BB,
        0x22D9C3, 0x0436ED, 0x650791, 0xA314EC, 0x42A0B3, 0x37A521,
```



```
        0x4BFB2B, 0x8C1B7F, 0x115E17, 0xF7C27F, 0xC1D5EB, 0x060487,
        0x8A28D6, 0x41330F, 0xBFAE67, 0x7774E8, 0x4CCC3C, 0x6B2F80,
        0x628BF2, 0x1E41A6, 0x8D0B22, 0xBC85BA, 0xCCF461, 0xBEC69C,
        0xDF8A10, 0x3C5E71, 0x2F8D5F, 0x63D3DA, 0x5934D1, 0x2CA35D,
        0xC687A2, 0x24E9B4, 0x1843D3, 0x5C9B97, 0x9B580C, 0x780B2C,
        0x59943D, 0x0744D0, 0x8DA6E3, 0x07AAF6, 0x2214D0, 0x72E8D7,
        0x54151B, 0x514DE9, 0x8DCC3B, 0x0CEB00, 0x2C4DE3, 0x5012AE,
        0xD7B72E, 0xB7DE9A, 0x641B2F, 0xF9CF17, 0x8BD282, 0x9F31A3,
        0xDED846, 0x467E05, 0x26CCEA, 0xF8E404, 0x65572E, 0x82C594,
        0xE572A9, 0x895653, 0xA1AA94, 0x8DD876, 0x5E9A61, 0x69EB1C,
        0x0385A9, 0x5BC844, 0x95B2DF, 0x6678F6, 0xFA7033, 0xE4F434,
        0x5584A9, 0x32C099, 0x9AD846, 0xB3FFD1, 0xA81C56, 0x4E54EF,
        0x54D173, 0xF191B4, 0x49B2A2, 0xB309D9, 0x546D8D, 0xC0A51E,
        0xCAFFC0, 0x785400, 0x05F69D, 0x894056, 0xC33098, 0xDFF6C2,
        0x908D97, 0x05CC96, 0x46484B, 0xBD7B9D, 0xB152F5, 0x5A7461,
        0x59CA20, 0x8F8EF5, 0xC9FF05, 0xF6F398, 0x856C97, 0x81E07C,
        0xAE5EDA, 0x51BDC9, 0xF26437, 0xBBC8CE, 0x091B52, 0x68B6A5,
        0x90750E, 0x925EF9, 0x3D9CB3, 0x46EA96, 0x97D648, 0x78BCC7,
        0xF4B488, 0x05275E, 0x6619DF, 0x56D4A0, 0x8C5C41, 0xDB345A,
        0x0B79DA, 0x496369, 0x96109B, 0x667664, 0xC40CF9, 0x91D7CA,
        0x119F1A, 0xA99272, 0xCBB529, 0xBB033E, 0x8F91C0, 0x570045,
        0xB845C2, 0x2B8E52, 0x687AFB, 0x0D0AA3, 0x200863, 0x043B83,
        0xF129DE, 0x49C2D6, 0x9641D2, 0xC4747C, 0x220804, 0x503F05,
        0x7E274F, 0xCA83D9, 0x9D6495, 0x0E5039,
};

#endif /* ! codereview */
const int _TBL_ipio2l_53[] = {
        0xA2F983, 0x6E4E44, 0x16F3C4, 0xEA69B5, 0xD3E131, 0x60E1D2,
        0xD7982A, 0xC031F5, 0xD67BCC, 0xFD1375, 0x60919B, 0x3FA0BB,
        0x612ABB, 0x714F9B, 0x03DA8A, 0xC05948, 0xD023F4, 0x5AFA37,
        0x51631D, 0xCD7A90, 0xC0474A, 0xF6A6F3, 0x1A52E1, 0x5C3927,
        0x3ADA45, 0x4E2DB5, 0x64E8C4, 0x274A5B, 0xB74ADC, 0x1E6591,
        0x2822BE, 0x4771F5, 0x12A63F, 0x83BD35, 0x2488CA, 0x1FE1BE,
        0x42C21A, 0x682569, 0x2AFB91, 0x68ADE1, 0x4A42E5, 0x9BE357,
        0xB79675, 0xCE998A, 0x83AF8B, 0xE645E6, 0xDF0789, 0x9E9747,
        0xAA15FF, 0x358C3F, 0xAF3141, 0x72A3F7, 0x2BF1D4, 0xF3AD96,
        0x7D759F, 0x257FCE, 0x29FB69, 0xB1B42C, 0xC32DE1, 0x8C0BBD,
        0x31EC2F, 0x942026, 0x85DCE7, 0x653FF3, 0x136FA7, 0x0D7A5F,
        0x93FC61, 0x035287, 0xC77FCA, 0x73530A, 0xC6BC15, 0x0E4B0F,
        0x568FCE, 0x2D3456, 0x4D7FE1, 0xA12CD1, 0xB2CEA2, 0x531C62,
        0x70B4D2, 0x1BCE9A, 0x87704D, 0x6B83D7, 0xAA8121, 0x2530EA,
        0x2074BF, 0x28A071, 0x9D69C3, 0x406DD8, 0xF58783, 0x115D89,
        0x5E85F3, 0xAACDCC, 0x8C0B57, 0xD7DFFE, 0x550D96, 0xC43EB4,
        0x89ABA7, 0x94F595, 0x56F260, 0x06A4CD, 0x7FD2E2, 0x6FDFA8,
        0x3E9C98, 0xBFD682, 0xAD3A12, 0x23A8A6, 0x173A89, 0x5DE9BD,
        0x95A978, 0x28E484, 0x5964F3, 0x496AF0, 0x4B1DA9, 0x989061,
        0xBD2BF2, 0xE01A90, 0x0905B7, 0xAC39AC, 0x52D5B7, 0x109F25,
        0x3AE1DC, 0xF90A7C, 0x33F4E5, 0xF5DFDF, 0x1522D0, 0x562CE6,
        0x392CFF, 0xEB9032, 0x10A08E, 0x0B1D7F, 0x42B80A, 0x366DD2,
        0xC24F89, 0x02222E, 0x21494C, 0x985287, 0x87FD07, 0x2EE361,
        0xAD8D68, 0xE72273, 0x9E8D59, 0xD09999, 0x10F4A1, 0x1079A3,
        0xE9BEAF, 0x9C0887, 0x09C622, 0xEBCF06, 0x974532, 0x086A8F,
        0x6CEA05, 0x388C00, 0x74969E, 0xC85B16, 0x385A38, 0x9A2F35,
        0x670531, 0xABA6D0, 0xEFD3C1, 0x27AD92, 0xF4203E, 0x3D619F,
        0x4D05F4, 0x9AE7CC, 0x03B592, 0x41FF55, 0xCAFCA5, 0x1A0987,
        0x88AB79, 0x3627D4, 0x25B12A, 0x52594A, 0xA2BEB0, 0x25C3F2,
        0x4489DA, 0x7959A7, 0xEAEC89, 0xB34714, 0x960196, 0x1FC33A,
        0x7F0275, 0x32EF92, 0x0111CE, 0x8E4685, 0x6F5B34, 0xF6123A,
        0x5543B2, 0xE9A02A, 0x74E03F, 0x54D5A8, 0x086A2C, 0x4A9CD3,
        0x921191, 0x229764, 0x0A1A84, 0x9B45AE, 0xC653A5, 0xB15F33,
        0x100FD1, 0x7DD740, 0xB20CD3, 0x0A0786, 0xF506C3, 0x25EBF4,
        0x3AB39E, 0xE3BB24, 0x27646F, 0xEECE57, 0x706BFE, 0xC7A869,
        0x57ED51, 0x118C82, 0x2B0FF5, 0xC8E545, 0xC43D80, 0x2A3183,
        0x4C1BB9, 0xBC108A, 0x099779, 0xF9ECC8, 0x2A1063, 0x5D2F6A,
        0x8F2675, 0x12FF6D, 0x32EED9, 0xE4A245, 0x7392CF, 0x5C240B,
```

```
        0xC476FF, 0x97AFC7, 0xB76131, 0x665E05, 0x67BD57, 0x19E998,
        0x3A5863, 0x23B8AA, 0x5B5608, 0x8A66C6, 0x5F2AD3, 0x78BAFA,
        0x3516CE, 0xCBEA16, 0x6E40D4, 0xB463D4, 0xA6C12F, 0xABD3D7,
        0x32650A, 0x579D10, 0x3CB9E2, 0x1A02A7, 0xDF2FFA, 0x28C991,
        0xB2264C, 0x027870, 0x47BDD4, 0xF243B1, 0x39AE2C, 0x282EA4,
        0xAF1D98, 0x2AFD16, 0xABE7AF, 0x17CB67, 0x8FF93E, 0x793167,
        0x435F6B, 0x48058B, 0x417DA0, 0xE01217, 0x085A69, 0xB50E36,
        0x79A4CD, 0xD74907, 0x26C4B5, 0xB90054, 0x06C3AD, 0x5AB38F,
        0x585E91, 0xD04E4F, 0x2938CE, 0xD4EAA7, 0xA06DE5, 0x40BFE5,
        0xDE6849, 0xEF65F0, 0xF1D4BB, 0x94C21E, 0x66E978, 0x1B9B94,
        0x961043, 0x5961B8, 0xBAAA74, 0xD662EE, 0x9DABF6, 0x0AFE28,
        0x9587A4, 0xA632BC, 0x09149F, 0xDEA996, 0x2CAFD7, 0xBDE29B,
        0x7159E6, 0x1F7C49, 0xF2E2ED, 0xBFA992, 0x7C77EF, 0xC245D0,
        0xB2D129, 0x993E75, 0xAB4C0C, 0x5C84B6, 0x17F542, 0x45314E,
        0x1DEF1B, 0xE3BDCC, 0xB3AE86, 0x24522F, 0x918FC6, 0x2138D5,
        0x883646, 0x6858B6, 0x032762, 0x5170F8, 0x4974EA, 0x76BF77,
        0xECDA8A, 0x9EADDD, 0x2404EF, 0xC52A5D, 0xF2E858, 0xC42D60,
        0xD18C08, 0xDE59B2, 0x4CC3A6, 0x94D888, 0x4C4AF0, 0xCF1F8C,
        0xBF2F6F, 0x7B4535, 0x98B0DB, 0x2BE0CF, 0x4616A7, 0xA8D9FB,
        0x88CA7A, 0x5087E1, 0x18DD8A, 0x1A9F4F, 0x1DCECE, 0xF8609E,
        0xE2F0C8, 0x9AD7D4, 0xE3CDFE, 0xC6FDD5, 0x8FF3CD, 0x7D45AA,
        0xD34957, 0x7C1963, 0x6CE098, 0xB70215, 0x326BBF, 0x47B3A6,
        0xF9235D, 0x6F66F5, 0xC6E40C, 0xE7F50B, 0xFF2FDD, 0x5A1251,
        0xE95EF1, 0xDE8E67, 0xECEE9B, 0xC9F98E, 0x722224, 0x6DF750,
        0x81D08F, 0x2BFCF0, 0xDDC10D, 0x775314, 0xDB1D87, 0x41626B,
        0x9EDF31, 0x7738D9, 0x8D9EB4, 0x4F1C2A, 0xF3E795, 0xB69699,
        0xD9A56D, 0x31BB1B, 0x542975, 0xAB917B, 0x63927C, 0x9BB764,
        0x84A598, 0x0A0C51, 0x5E48C4, 0x7780E3, 0x87E156, 0x155972,
        0xE406F8, 0x48AB9E, 0x3CCDDA, 0x010F87, 0x683B70, 0x400CAD,
        0x5DE5C5, 0x7262FA, 0xFA248D, 0x013AF2, 0xE2E8B5, 0x995F7D,
        0x7F8C4B, 0x0E8B59, 0x1006F1, 0x40B6E9, 0x760654, 0xCBCC8C,
        0x086F40, 0xDC7F6F, 0xFCD0D4, 0xA47ADE, 0x5204FA, 0xF38A9D,
        0xE76C7C, 0x575207, 0x499BF1, 0x0DB01C, 0x09098E, 0x957A71,
        0xD53E0E, 0x61DF1D, 0xE6EF34, 0x5821EC, 0x96BCC0, 0xDC96CE,
        0xA9C0AE, 0x130B2C, 0xCCC589, 0x829BB9, 0x2A75BA, 0x97611C,
        0x0CEAB8, 0x165D9D, 0x35AD41, 0x82A805, 0x975628, 0x5601A6,
        0x074F08, 0x80A27D, 0xEFA64E, 0xD7BB4B, 0x5E6397, 0xC92FFC,
        0x4F3F7A, 0xBEA764, 0x0C9B7D, 0xC5DC74, 0xEAD216, 0x6DBBC0,
        0x913E3E, 0xABF50B, 0x95B24A, 0x3FC9C5, 0xE7BA15, 0x8C7F70,
        0xF81358, 0x774606, 0xCE8C0D, 0xB6B268, 0xB85BA6, 0xAC9B2E,
        0x1AAB05, 0x0C6C82, 0x6EC2AE, 0x606874, 0x8F60BF, 0x1FBC7B,
        0x58C97A, 0x448794, 0xBA48A0, 0x72E882, 0x6D3568, 0xE131FD,
        0x4745D0, 0x0BFA1E, 0x07B01D, 0x474D43, 0x59387E, 0x5B0AD5,
        0xC37A8C, 0x0474E8, 0x13D99D, 0x68A13C, 0xB69118, 0x89228C,
        0x6F7D83, 0x86D665, 0x5C7744, 0xDD183E, 0x1C2E17, 0x712F5E,
        0x4AACCB, 0xB69B68, 0xA1201F, 0x743C2B, 0xF6AD70, 0x92E024,
        0xF34FD8, 0x33712E, 0xFE1D73, 0x4471F0, 0x7D0526, 0x58AF47,
        0x7B11FE, 0x1FCE4F, 0x1356C9, 0x9CE3CA, 0xA843C0, 0x8EEA3C,
        0xABEEE4, 0xA5D495, 0xA407A4, 0x31BB4B, 0x0AA1E3, 0x518E7C,
        0xAA4A66, 0xD82CD8, 0x6EF8D2, 0x6F32E6, 0x1DC26B, 0x17AE59,
        0x4B683B, 0x8D48F7, 0xF4FBD8, 0xD4FE0A, 0xE961DE, 0x87BD37,
        0xE6CCD6, 0xCBD76D, 0x3E99DE, 0xB72E21, 0x54EB90, 0x6AB45D,
        0x600AFB, 0xA17B2F, 0xDA0421, 0xE6CA95, 0x35AAA2, 0x7D8FB1,
        0x3207BB, 0xBF82EE, 0x71F55F, 0xC661CB, 0xBD72A1, 0xBF5A64,
        0x6E39E8, 0x6C6DE2, 0x2BD178, 0xAF62A5, 0xA7D86E, 0xE7D0FE,
        0x84DB03, 0x67FDA2, 0x2D6809, 0x0F8B8F, 0x1B50E3, 0x234EF5,
        0x7325ED, 0x8F8F4C, 0xC1E426, 0x3066AD, 0x0759A4, 0xE03390,
        0x70CC9A, 0x524F77, 0xCDD489, 0x97DD24, 0xA81858, 0xF24513,
        0xA9C18E, 0x2A2F82, 0xC2C014, 0xB8E7F0, 0x934036, 0xD36E51,
        0xD9A089, 0xDBC587, 0xB30418, 0x969192, 0x0A5213, 0xE21841,
        0x2881EC, 0x9A293F, 0x0DF705, 0x85B497, 0xE430B9, 0xE90ECF,
        0xC15FDC, 0x9E8A7E, 0xC5472D, 0xB54FBD, 0x456AF2, 0xCA80B6,
        0xAE25FE, 0xA03B46, 0x6C6CFD, 0x78382A, 0x0E7877, 0x7F2D31,
        0x03C827, 0x61CF52, 0x339A2F, 0x2286A9, 0xE41DF0, 0x640F5C,
        0xBEF364, 0x010506, 0x6D2C21, 0x841EFF, 0x7F3B5D, 0xD98DC8,
        0x0F9421, 0xA25B0C, 0x4C2C44, 0x922392, 0xB98A8A, 0x6179B9,
```



```
        0xF7B419, 0x289AAF, 0xE92F47, 0x5E47A2, 0x82927F, 0xC7290E,
        0x6C925C, 0xBA5A3C, 0x8FB7F6, 0x9C4BEE, 0x02C529, 0x0CFCD7,
        0x5EBD8C, 0x7196E0, 0x4B917E, 0x6B9780, 0x6A1731, 0xA617FF,
        0x27A20D, 0x5A56A3, 0x43C4DB, 0xC62EA4, 0x637A84, 0x1C46F9,
        0x33C780, 0x61A278, 0x4915C9, 0xD6C776, 0x6A7C66, 0xD8DD0C,
        0xF87EB1, 0x124C43, 0x5B87E7, 0x097456, 0x3C2FA7, 0x307C4A,
        0x54267A, 0x30E34E, 0xC0CF98, 0xD75B19, 0xFADEDB, 0x12CBE8,
        0x29F24C, 0x579C7E, 0xBF3682, 0xDCB460, 0xAE08B3, 0xA524BC,
        0xC181C2, 0x5DAB90, 0x466602, 0x55345B, 0xA13941, 0x47D820,
        0x278066, 0x81B089, 0x165EFB, 0x4D27FD, 0x2BF9F4, 0x2E2FFB,
        0x6106B5, 0xE76806, 0x445A84, 0x0BDA0D, 0x49D7A4, 0x72650D,
        0xCDC55B, 0x3E16BC, 0x132F6F, 0x29E8FD, 0xE58428, 0x621E41,
        0x7D2AC4, 0xAB5697, 0xAC61EB, 0xE5DAF0, 0x654ED6, 0x8E77E3,
        0x0B2FBC, 0x2E63A3, 0xC8296A, 0x8B631F, 0x4ECCA6, 0x91859C,
        0x9E3E45, 0x0E3CC7, 0xC12454, 0xCCBCB6, 0x17979E, 0xD0D374,
        0xA489A2, 0xC6258F, 0xE8EF9E, 0x12EE26, 0xC614C2, 0x62E23E,
        0xCA8C5C, 0x409AC9, 0x511D05, 0xA88CE0, 0x195500, 0xF7144F,
        0x913BB7, 0x17D064, 0xF6C9CE, 0xAC5D11, 0xD0C313, 0xBCCCB6,
        0xAAD4FC, 0xE47B2C, 0xFE4362, 0xF2E712, 0x2D5EFF, 0x833822,
        0x58A1D7, 0x68377C, 0xE49B25, 0x22B179, 0x048796, 0x069400,
        0xE670D3, 0xD2CB85, 0x55FBE6, 0x67F281, 0xFE2DE0, 0x8CFAF2,
        0x9865BC, 0x210CD3, 0x86DD70, 0x43D00F, 0x55E279, 0x679252,
        0x8D4F58, 0xE17AC5, 0x6A6127, 0x1B0876, 0x5D8ED0, 0x701330,
        0xD5BD25, 0xC9A126, 0x57C571, 0xDC5C3F, 0xB6D34E, 0xB72383,
        0x001A9E, 0x7D36C0, 0x8151F6, 0x65D7C1, 0xE1F513, 0xCD372A,
        0xE69B0C, 0xD02685, 0x23C3EB, 0x3544CB, 0xF0BE31, 0x83F399,
        0xCB93F8, 0xFFC693, 0x908EC6, 0x8E5DE1, 0x315B7E, 0x67CE7B,
        0x40AAF7, 0x7FD285, 0x069B36, 0x03C00A, 0x13C7D5, 0x0DA14C,
        0x1EAAD4, 0x2B777F, 0x8E05C1, 0x5AD1AE, 0x60C398, 0xA4EA59,
        0x10BEED, 0x88F2FA, 0x69B941, 0xA54E70, 0xA817C3, 0xB96246,
        0xE8EEDC, 0x56D570, 0xBBEBB5, 0xD8F235, 0x201AB9, 0x9CC747,
        0x5BC2FB, 0xC877F3, 0x428CF6, 0x4EEF84, 0xBF85FD, 0xEE6D34,
        0x84C2DE, 0xC42F4C, 0x1A513B, 0x9AC41F, 0x87FFFA, 0x1CA431,
        0x714252, 0xC73FB9, 0x662D89, 0x3D83BA, 0xBDF046, 0x2E4F62,
        0x76B7C0, 0x81336C, 0xBE80A9, 0x4C9D72, 0x739A15, 0x47972C,
        0xA36A1B, 0xD31731, 0x54BA46, 0x2E8C72, 0xFEA5A5, 0x9A7E5F,
        0xC359ED, 0x8F0FFB, 0x1270DA, 0x5E9B08, 0xB0BFCB, 0x36974C,
        0x6CD8F9, 0xD02E1F, 0x1C3F2F, 0xFCF8F0, 0x4C2C6D, 0x0B2169,
        0x48B9CE, 0x42737D, 0xA8E974, 0x64062D, 0xA86C59, 0xEEC419,
        0x047C83, 0x996A23, 0xF2A4C8, 0x4BE1B8, 0x348286, 0xE84240,
        0x8337CB, 0xE55A2F, 0xC17750, 0xA4DA06, 0x64347F, 0x59A5A1,
        0xDFF53D, 0x62A571, 0xEECF3A, 0x886700, 0xC06DAF, 0x4E161F,
        0x12670E, 0xBDFE1A, 0xA72B38, 0x5BA22C, 0xFED227, 0x3FC814,
        0x150E5A, 0xE99B3A, 0x8EE9FC, 0xBC1845, 0x32373A, 0xBDA476,
        0xCEB88F, 0x7FAED3, 0xDB9116, 0x31CF72, 0x1A5136, 0xC4F362,
        0xDE4799, 0x768043, 0x386207, 0x8E5497, 0xB0EF6D, 0x6C57FB,
        0xF56664, 0xD24F05, 0xE0F702, 0x8A41EF, 0xA2EC53, 0x09731C,
        0x6157FE, 0xC5731C, 0xEF1A2E, 0x60EC10, 0xA67EFE, 0x486A73,
        0x8004F6, 0xC3F482, 0x63BA28, 0x107282,
};
#endif
```

```
**********************************************************
    7823 Thu Jun  6 20:38:29 2019
new/usr/src/lib/libm/common/Q/_TBL_logl.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

/*
 * Table of constants for logl.
 * By K.C. Ng, March 9, 1989
 */

#include "libm.h"

const long double _TBL_logl_hi[] = {
        +0.0000000000000000000000000000000000000e+0000L,
        +1.5504186535965254150854044093203958752 74e-0002L,
        +3.0771658666753688371028204543134235494 27e-0002L,
        +4.5809536031294203166679264495252313016 34e-0002L,
        +6.0624621816434842580606129724837429974 42e-0002L,
        +7.5223421237587525698605320390868275788 24e-0002L,
        +8.9612158689687132619951437305287870275 78e-0002L,
        +1.0379679368164356482606176948034383482 44e-0001L,
        +1.1778303565638345453879405815045481685 63e-0001L,
        +1.3157635778871927258871606243121355966 98e-0001L,
        +1.4518200984449789728193505722861831238 87e-0001L,
        +1.5860503017663858409337116835300529813 49e-0001L,
        +1.7185025692665922234009888125168925235 99e-0001L,
        +1.8492233849401199266390350597230773146 15e-0001L,
        +1.9782574332991988036257203745337829257 63e-0001L,
        +2.1056476910734963766955275316260887024 97e-0001L,
        +2.2314355131420975576629499370279971040 32e-0001L,
        +2.3556607131276690907758811630090963371 26e-0001L,
        +2.4783616390458125678060268679527200816 99e-0001L,
        +2.5995752443692606697207949330852317390 65e-0001L,
        +2.7193371548364175883166942420311366852 88e-0001L,
        +2.8376817313064459834690104587945471353 12e-0001L,
        +2.9546421289383587638668177009161075350 02e-0001L,
        +3.0702503529491186207512436665524198343 11e-0001L,
        +3.1845373111853461581024711406860785615 31e-0001L,
        +3.2975328637246798181442280143624787844 88e-0001L,
        +3.4092658697059321030508900015446624269 52e-0001L,
        +3.5197642315717818465544745520482542881 30e-0001L,
        +3.6290549368936845313782429453982725215 23e-0001L,
        +3.7371640979358408082101673312267575254 99e-0001L,
        +3.8441169891033203973479004873695081103 20e-0001L,
        +3.9499380824086897810639397835206553189 15e-0001L,
        +4.0546510810816438197801303229941379322 04e-0001L,
        +4.1582789514371096561332882595117448266 05e-0001L,
        +4.2608439531090006312454473855698423563 71e-0001L,
        +4.3623676677491807034904122391785095757 36e-0001L,
        +4.4628710262841951153258998740559942080 63e-0001L,
        +4.5623743348158759438080538408187057197 56e-0001L,
        +4.6608972992459922455861918787364536547 69e-0001L,
        +4.7584590486996391426520938936776558244 36e-0001L,
        +4.8550781578170080780179106330112555350 46e-0001L,
        +4.9507726679785151459796452130348994806 04e-0001L,
        +5.0455601075239528705830818288179488164 63e-0001L,
        +5.1394575110223431680100586682876695249 12e-0001L,
        +5.2324814376454783651680693535350375638 40e-0001L,
        +5.3246479886947184387392343795832631511 44e-0001L,
        +5.4159728243274437157654221116898413568 47e-0001L,
        +5.5064711795266227925994798613045553648 07e-0001L,
        +5.5961578793542268627088834665328436032 87e-0001L,
        +5.6850473535266871207873858040829459937 34e-0001L,
        +5.7731536503482360431811170675594990732 34e-0001L,
        +5.8604904500357820890411939164020353168 40e-0001L,
        +5.9470710774669278951434349590056581348 79e-0001L,
        +6.0329085143808426234058496635521551666 82e-0001L,
        +6.1180154110599290352988976088821255236 26e-0001L,
        +6.2024040975185752885149429543236279432 83e-0001L,
        +6.2860865942237413774430812939979007275 20e-0001L,
        +6.3690746223706923162049422813721571230 62e-0001L,
        +6.4513796137358470166522829833408641609 16e-0001L,
        +6.5330127201274563875861571909468580139 03e-0001L,
        +6.6139848224536500826023544877769330609 28e-0001L,
        +6.6943065394262926729888508450595770033 79e-0001L,
        +6.7739882359180614080968245650252746174 92e-0001L,
        +6.8530400309891941654404766999569518506 29e-0001L,
};

const long double _TBL_logl_lo[] = {
        +0.0000000000000000000000000000000000000e+0000L,
        +1.9492428771251263890303741483552770373 60e-0027L,
        +3.0536379285974251562898000588306290523 49e-0027L,
        +3.1194110290975925497245992197965796983 55e-0027L,
        +2.3155828333117796947293020298740440047 47e-0027L,
        +1.9592794138848629196962306424810016449 14e-0026L,
        +3.2073196658509406891125909313215845852 32e-0026L,
        +3.4283630853482158869012402005606901914 23e-0026L,
        +5.1320066888212186442797930356391585911 04e-0026L,
        +6.6258269602781916230613139029871366756 70e-0026L,
        +6.5119460111338299044782139989273808177 16e-0027L,
        +6.2728362771108058770481262335487100958 28e-0027L,
        +6.4803458012575463263114230100011848013 74e-0026L,
        +8.6686941889544302560184918553370055162 55e-0026L,
        +3.3743583176498967535335829211308009233 37e-0026L,
        +5.9569742643470821864292479445186677575 30e-0026L,
        +9.6607034792971448649414617855651801914 97e-0026L,
        +1.0264013377642437288559586071278317182 21e-0025L,
        +7.8951252739829033515418225476253519740 82e-0026L,
        +1.2337878706698339852746113295316527538 61e-0027L,
        +7.0329885493453776747364244782965163775 26e-0026L,
```

```
        +1.764708000531295728633384847670848625081e-0025L,
        +1.359633534416813878749988462341486606257e-0025L,
        +1.738801359182578816100029030519562527565e-0025L,
        +9.952199173944211463901058384143333287734e-0026L,
        +1.048454193250289008158931842356333139861e-0026L,
        +1.996258899657478647716755914664160562170e-0025L,
        +1.054613497176328160439100383508915283893e-0027L,
        +5.143766259398803158035428613944687700657e-0026L,
        +9.959314775409457843445608446369116918733e-0026L,
        +1.374434005748650164937032849496159512584e-0026L,
        +5.801291623641845255360276144691829323298e-0026L,
        +8.316493534335158882618991007102844149005e-0026L,
        +6.700372782269538472749252935215499311080e-0026L,
        +1.410384923832595967313936024639114199873e-0025L,
        +9.914327034309007140234547094345224044051e-0026L,
        +1.932140695859428972988292357113036038299e-0025L,
        +1.540820591764623257530922632028001952748e-0025L,
        +5.963112403438125368118769047371614538741e-0026L,
        +1.969366158297316138140115855981754471320e-0025L,
        +1.388966334707414023926476567157219393213e-0026L,
        +6.353934371729676603785277612987160899450e-0026L,
        +3.488563800483361999633395030516586696799e-0025L,
        +2.214454506406188993139159148705861598088e-0025L,
        +2.895813670852564643073769701905380524878e-0025L,
        +2.855018159274929532107406110765900047355e-0025L,
        +9.273144996328510392949991151883397780965 8e-0026L,
        +1.930744579236138780895942105787011752697e-0025L,
        +1.538735422331574088102192677519746877453e-0025L,
        +1.844586676642028985383989272409206538468e-0025L,
        +3.547635464941839708071563131885310128521e-0025L,
        +4.464712081783102087084281748635332222581e-0026L,
        +5.062863951970459495500575300347508324877e-0026L,
        +2.203060950889790157204518257910818074191e-0025L,
        +5.540602231323196163388428517126435254723e-0027L,
        +3.371348840624439923830692211721531149909e-0025L,
        +7.637439356719457811667844141793488670929e-0026L,
        +1.990439834788842292780211676828666657547e-0025L,
        +1.978006454898465493718923085569873769719e-0025L,
        +1.621161880831806223416081355472819612309e-0025L,
        +3.899319576320551292151632804501913965920e-0025L,
        +1.864235278097858865893177670582100390924e-0025L,
        +1.534948208368053655735541548539936152221e-0025L,
        +4.089715378013580174759550633443176148182e-0025L,
};
```

```
**********************************************************
    9105 Thu Jun  6 20:38:29 2019
new/usr/src/lib/libm/common/Q/_TBL_sinl.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

/*
 * table of sinl(x) where x is 0.15625 + i*0.03125, i=0,1,...,74.
 * {0x3ffc4000,0,0,0} --> (inc 0x800) -->  {0x3ffe9000,0,0,0}
 *    0.15625              0.03125           0.78125  (pi/4 = 0.785395663...)
 */

#include "libm.h"

const long double _TBL_sinl_hi[] = {
        +1.556149927735560412099206432035162581492e-0001L,
        +1.594724589318434199425963881130908091043e-0001L,
        +1.633274917366128508468661724543543700180e-0001L,
        +1.671800323648067343709660282007512722777e-0001L,
        +1.710300220313950192813479692398343312832e-0001L,
        +1.748774019902721898956853691085201901772e-0001L,
        +1.787221135351536593753562418641807235164e-0001L,
        +1.825640980004715553995456513594130154574e-0001L,
        +1.864032967622698845523799831032052611919e-0001L,
        +1.902396512390990617639858876307573287214e-0001L,
        +1.940731028929097911560552002141454036336e-0001L,
        +1.979035932299462846523939109918127853182e-0001L,
        +2.017310638016388047250381511640009707423e-0001L,
        +2.055554562054955176568330206054936963632e-0001L,
        +2.093767120859936437118907527248816522107e-0001L,
        +2.131947731354698906160730331184784624987e-0001L,
        +2.170095810950101567605780958260553963420e-0001L,
        +2.208210777553384905528563479277490523429e-0001L,
        +2.246292049577052923504285497964248198189e-0001L,
        +2.284339045947747454247378313461956799859e-0001L,
        +2.322351186115114624139308777462358722636e-0001L,
```



```
        +2.360327890060663337354342917945180835158e-0001L,
        +2.398268578306615644413692518108865737937e-0001L,
        +2.436172671924748860122309477052146367777e-0001L,
        +2.474039592545229295968487048493892032583e-0001L,
        +2.549659604158784674875565748648726276685e-0001L,
        +2.625123997691532814509496263956929310415e-0001L,
        +2.700428167185850315527550636188270542366e-0001L,
        +2.775567516463363259220234468281285678680e-0001L,
        +2.850537459405474245877630333232525606110e-0001L,
        +2.925333420233275436247023264939134225079e-0001L,
        +2.999950833786830511632482820116999437532e-0001L,
        +3.074385145803808506705029582019820907725e-0001L,
        +3.148631813197452508650363151269390156066e-0001L,
        +3.222686304333866256877459198931880313050e-0001L,
        +3.296544099308601719143177251264631756945e-0001L,
        +3.370200690222530762612817541738100244419e-0001L,
        +3.443651581456984082071720464722237468910e-0001L,
        +3.516892289948140592225848969555470155541e-0001L,
        +3.589918345460650536777102991528689411936e-0001L,
        +3.662725290860475613729093517162641768533e-0001L,
        +3.735308682386929464168397526608481120900e-0001L,
        +3.807664089239019205720073388896675081 3e-0001L,
        +3.879787097270250460510796908137419597834e-0001L,
        +3.951673302409342362448326404196536570776e-0001L,
        +4.023318317777731112171055988809823868862e-0001L,
        +4.094717770532950661226940270114522362676e-0001L,
        +4.165867302820411192591124488310696565000e-0001L,
        +4.236762572039380103616839880311024798208e-0001L,
        +4.307399251108031972163215178508491897943e-0001L,
        +4.377773028727551328616189747027966801523e-0001L,
        +4.447879609645272114330560125295252111499e-0001L,
        +4.517714714916837765816887501340628695303e-0001L,
        +4.587274082167365923772950289728747732442e-0001L,
        +4.656553465851601826811995125075467791328e-0001L,
        +4.725548637513044511465513178085169418350e-0001L,
        +4.794255386042030002732879352155714019245e-0001L,
        +4.930786857539230572651365527534871205832e-0001L,
        +5.066114548142573676422960008938671919466e-0001L,
        +5.200205419537270047602136998746747297451e-0001L,
        +5.333026735360201733291311033081615288994e-0001L,
        +5.464546069192035644033495537494110008818e-0001L,
        +5.594731312473668773848440060031166884132e-0001L,
        +5.723550682345072403849537068245036075406e-0001L,
        +5.850972729404621548053993141500804585059e-0001L,
        +5.976966345387015312386476189673343370299e-0001L,
        +6.101500770757913712737423935661832200218e-0001L,
        +6.224545602223436830419267050904433302049e-0001L,
        +6.346070800152692968503099142036714364826e-0001L,
        +6.466046695911523705240421598828007629792e-0001L,
        +6.584443999105675415895839548840419894659e-0001L,
        +6.701233804731628946545315835006484946172e-0001L,
        +6.816387600233341667332419527798939078545e-0001L,
        +6.929877272463179102818154908230482095679e-0001L,
        +7.041675114545336727800595099739428438828e-0001L,
};

const long double _TBL_sinl_lo[] = {
        -7.839895634192879801217180506294972695887e-0036L,
        -7.579278167533093253112813720340914585189e-0036L,
        +1.813803443011554857703679023007542917336e-0036L,
        -5.685040200337201343842157163322014327778e-0036L,
        +7.013958751874876088754160302032414326691e-0036L,
        +9.101164084055805006113433827277389417722e-0036L,
        -1.529069265172651032025475612605940319203e-0036L,
        -5.873100812266872079952884219254900231461e-0036L,
        +1.764603048068267800105867159753318395454e-0036L,
```

```
        +1.747799267790272859521729635868399475234e-0036L,
        -9.673047410519982672089452429449289994858e-0036L,
        -7.666827750837122707923169727244402427704e-0036L,
        -4.275134347549669784351512906173841196088e-0036L,
        -1.826904072780322152815985026139121969706e-0036L,
        -1.594702873443294499653146384825158092559e-0036L,
        -7.180615084240582786256765419723871383233e-0036L,
        +1.073564887942168318128295491982011935257e-0035L,
        +6.166267602604185314123111207543917974633e-0036L,
        +2.420615108492974698446957518700585915995e-0036L,
        +1.864291640707538541155008952901532832506e-0036L,
        -4.969304833641910200750246243329289676583e-0036L,
        +7.191910920600591837788283739445222790835e-0036L,
        +2.398670365698962872409384444450714480056e-0036L,
        +2.625717623049256499265563616201152710192e-0036L,
        -7.364870011085995329435971152758116180239e-0036L,
        +2.202803779185347210050716883280741537850e-0035L,
        +3.249236770720310646731771785718217268891e-0036L,
        +2.438735936561976529428558055804286674772e-0035L,
        -1.358485954689981282143446687700830546868e-0036L,
        +2.042693258859029188027001236804037487674e-0035L,
        +1.935394086687044503080036879506851279569e-0035L,
        +1.351742655356978501392833614755710504356e-0035L,
        +1.065151724232046458392410994534171402266e-0035L,
        +1.924312402124329269930577050628341603064e-0035L,
        -1.495058978047592634838539083355002279867e-0035L,
        -1.226069967847432149730821922942328537678e-0035L,
        -2.214357561488394736777775450498906642993e-0035L,
        -3.197918850054809249377584675940519273161e-0036L,
        +1.752934334182702105675254128020832940341e-0035L,
        -2.067723892627233681394169702571120887364e-0035L,
        -1.967684335349365926758978182531089889151e-0035L,
        -1.480234947789865560488791134115171284680e-0035L,
        -2.020095411752086363369245333724961071903e-0035L,
        +8.019047838709350758444432786175864173856e-0036L,
        +7.575600313883125509400401940426278198665e-0036L,
        -1.956787228828481747235699165048715626458e-0035L,
        +2.239452414684575979216557857298213538383e-0035L,
        -2.004881068319988136754382697969636119420e-0035L,
        +1.404844563886544703294730965793125947043e-0035L,
        +1.540967800016293988508912183967615475673e-0035L,
        +9.627943645034426124771174260339225827341e-0036L,
        -1.671879365114936780075083716139548989818e-0035L,
        -1.193872230164722958937943872752845047434e-0035L,
        -4.709469941941829089292517195754317215227e-0036L,
        -1.562825989789718724786197721553059612264e-0035L,
        +9.313247749577680188502242676253713195205e-0036L,
        -1.384269776167183189501758486393819264119e-0035L,
        +7.064986931125350563523011010886249504328e-0036L,
        -3.109636998242741557027060430659670849804e-0035L,
        -3.324150213308849248337118428668967104680e-0035L,
        -3.427152913195516159969937952267551337396e-0035L,
        -2.118702307301603154209365237718648576399e-0035L,
        -1.289226205241639223068869521009917813361e-0037L,
        +2.125722734799331239445801994645145285587e-0035L,
        -1.781645762780561951365253354033804640300e-0035L,
        -2.495276089408737145274279413504615537138e-0035L,
        +1.338422379299389637809694183691505317685e-0035L,
        +1.919747869211470727176212361922698586017e-0035L,
        +4.330259169399683693260601564559270596091e-0036L,
        -3.417429818162194124156743659460798263758e-0035L,
        -4.544129440843003305237213918657872189520e-0035L,
        -1.331658529527437298976340693936847286647e-0036L,
        +2.748387759350275490242241143386673942983e-0035L,
        +4.500898710776635571808492195291899181897e-0035L,
        -2.872593727403933486766336102755986165044e-0035L,
```



```
};
```

```
**********************************************************
    9105 Thu Jun  6 20:38:30 2019
new/usr/src/lib/libm/common/Q/_TBL_tanl.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

/*
 * table of tanl(x) where x is 0.15625 + i*0.03125, i=0,1,...,74.
 * {0x3ffc4000,0,0,0} --> (inc 0x800) -->  {0x3ffe9000,0,0,0}
 *    0.15625              0.03125           0.78125  (pi/4 = 0.785395663...)
 */

#include "libm.h"

const long double _TBL_tanl_hi[] = {
        +1.5753410732527161068522577417241048648 70e-0001L,
        +1.6153978404952147630927524001104639774 18e-0001L,
        +1.6555051927393397621393091258505239004 70e-0001L,
        +1.6956644521976651015097060654375001942 64e-0001L,
        +1.7358769476798152084467341143536163299 85e-0001L,
        +1.7761440147744672763174292695868822438 19e-0001L,
        +1.8164669960332142765827589617435358648 82e-0001L,
        +1.8568472411563441162666122786498650671 49e-0001L,
        +1.8972861071805913289108337007303380698 29e-0001L,
        +1.9377849586689186351602239776826947804 40e-0001L,
        +1.9783451679023866880840637512397974093 03e-0001L,
        +2.0189681150741713288406899336576667577 69e-0001L,
        +2.0596551884857887210873932880303586088 78e-0001L,
        +2.1004077847455898084151752322459118625 45e-0001L,
        +2.1412273089695866488606681415862468386 3e-0001L,
        +2.1821151749846743250588204814957960843 82e-0001L,
        +2.2230728055343133087228881758799958296 92e-0001L,
        +2.2641016324867383747767140450355950999 74e-0001L,
        +2.3052030970457614145544751553791817539 38e-0001L,
        +2.3463786499642367899936771056107702682 68e-0001L,
        +2.3876297517602592026815106374093992765 66e-0001L,
```



```
        +2.4289578729361654240108594301566098811 74e-0001L,
        +2.4703644942004126466349473251580352721 70e-0001L,
        +2.5118511066924076739910389067743442152 46e-0001L,
        +2.5534192122103626650448223649047369079 38e-0001L,
        +2.6368059641999679985482599487946799896 58e-0001L,
        +2.7205369865877088342656436677127272204 98e-0001L,
        +2.8046247014525140316960428918526502560 07e-0001L,
        +2.8890817244051472600158844546424481636 30e-0001L,
        +2.9739208726902458946719401602465549007 16e-0001L,
        +3.0591551735305926410723892319699295799 42e-0001L,
        +3.1447978727257151617343822022562722570 22e-0001L,
        +3.2308624435174552011830065571796198670 07e-0001L,
        +3.3173625957357276733942970301053343756 85e-0001L,
        +3.4043122852383038742822744189025876874 99e-0001L,
        +3.4917257236591035225471296368439122105 18e-0001L,
        +3.5796173884801699838839596317944711797 52e-0001L,
        +3.6680020334432342272060481855376613597 12e-0001L,
        +3.7568946993175484040924577568759772548 06e-0001L,
        +3.8463107250414922304085627965828165062 83e-0001L,
        +3.9362657592563275822941378710121807798 93e-0001L,
        +4.0267757722514021177859373599000672509 49e-0001L,
        +4.1178570683410847578884987638487124158 95e-0001L,
        +4.2095262986947582207479414145067394718 50e-0001L,
        +4.3018004746423004902032960544727524433 02e-0001L,
        +4.3946969814786624048366314847993095103 27e-0001L,
        +4.4882335927923970884055552392457403316 72e-0001L,
        +4.5824284853443236695918919652415671847 90e-0001L,
        +4.6773002545239179992134270696199262296 70e-0001L,
        +4.7728679304125226172240425901042373553 91e-0001L,
        +4.8691509944840632449817175035683195875 449e-0001L,
        +4.9661693969756562571056057907256932001 64e-0001L,
        +5.0639435749622981207082275470717716019 70e-0001L,
        +5.1624944711717514449177533793692869114 20e-0001L,
        +5.2618435357779144170613437951009367774 4e-0001L,
        +5.3620128358121603134757892923930831268 26e-0001L,
        +5.4630248984379051325517946578028535441 47e-0001L,
        +5.6676706558058644568017794413547599907 92e-0001L,
        +5.8759736759144322139415886315789768952 06e-0001L,
        +6.0881374032438072141249397439637682349 39e-0001L,
        +6.3043767383588476685261142929977517401 01e-0001L,
        +6.5249189792880799272389773655162674722 27e-0001L,
        +6.7500048514424290766317794947772287205 41e-0001L,
        +6.9798896362359925514976572339001365161 19e-0001L,
        +7.2148444099090441998951788327959466390 42e-0001L,
        +7.4551574055939199513613016467781378046 17e-0001L,
        +7.7011355134420870500598366005277319752 10e-0001L,
        +7.9531059356867418564560169174741835480 89e-0001L,
        +8.2114180158989412191142396537471174252 36e-0001L,
        +8.4764452644655265409078830880761872355 13e-0001L,
        +8.7485876055448234953967190793215555721 47e-0001L,
        +9.0282738745267350219617436525397632084 64e-0001L,
        +9.3159645994407246116520275657393640746 20e-0001L,
        +9.6121551049437041618530062594687352673 85e-0001L,
        +9.9173789836326868025685738992990235605 95e-0001L,
};

const long double _TBL_tanl_lo[] = {
        +4.1792143859766888492509792029726635420 33e-0036L,
        +1.2015284461910252468390246502983979025 79e-0035L,
        +1.1293234894495377380809017887562319773 00e-0035L,
        +2.1401352789649361258155817582676490331 36e-0037L,
        +4.4322057493001850010408194569888626849 51e-0036L,
        +6.1361009781201322713326842071007406799 06e-0036L,
        -1.0325530595791808499873958321569766137 65e-0035L,
        -3.1600242599224370012158514041966523768 71e-0037L,
        +9.2880625289884281909637918183360249138 81e-0036L,
```

```
        -7.4469716957906447075469431193541677216 12e-0036L,
        -3.1941154067656331712329612143851010742 52e-0036L,
        +8.6368241010002712959254872128337700930 90e-0036L,
        +3.1022723672615915298582208844135843035 0e-0036L,
        -5.8519064735893686944872024417180089097 53e-0036L,
        +4.0100220701373069253385045978973360026 13e-0036L,
        +1.0377277068846733938759708743734621943 21e-0035L,
        -7.3732348604210605050990333196016580819 63e-0037L,
        +1.0125641876152431788993249433426629087 33e-0035L,
        -1.4093727127258765536015555741394389390 44e-0036L,
        +8.3788270249227671513628823098346454481 53e-0036L,
        +2.9738244784677708776774656460134774932 11e-0037L,
        +5.4000993987839063702709198488392765750 83e-0036L,
        -6.4625122424584154982627233249733886583 84e-0036L,
        -2.3227620230613189257505036425710134659 85e-0035L,
        -1.2589558871711939545565215792152598476 92e-0035L,
        -2.3204479558051791545213334959995649058 99e-0035L,
        -1.1490125523453291938344375580814843460 41e-0035L,
        +1.4527518178711698336239450313119443938 71e-0035L,
        +1.2335204198846725191888496884988149531 15e-0035L,
        -2.8017160589195629915001892194644566184 91e-0036L,
        -8.6523105517106080966339926122701875379 21e-0036L,
        +1.2471727167484077725648311284018808470 54e-0035L,
        -1.2397042496389302135839292473140245608 61e-0035L,
        +5.1844629880686161682338162965291506447 37e-0036L,
        -6.8564767234153913058575310957444425235 49e-0039L,
        -9.7395535312954336733984543443150390022 45e-0036L,
        +2.2662330164926606616382921267774015383 48e-0035L,
        +2.3015027700523766283479236217045621217 97e-0035L,
        +1.9488457473360570515383180074421149957 44e-0035L,
        -1.9407503893356082593633263705569144752 78e-0035L,
        +2.0196446608734582151184831630763147031 63e-0035L,
        +1.6020158121569059148212088070830629845 50e-0035L,
        -3.2924163925157433747432365078065462844 38e-0036L,
        +8.6638139423516724903283812713917042830 86e-0036L,
        +2.3666095815065990840939102172779947368 71e-0035L,
        -1.4089500631010566440399008540577765966 20e-0035L,
        -1.5147699209628490770131139236038035734 45e-0035L,
        -2.2619737955986151054494624430443300739 03e-0035L,
        -2.5532118821724020688664293900719809231 44e-0036L,
        +1.4166270294371260896759982573350333821 40e-0035L,
        +2.3427249317142491335892300798098508692 66e-0035L,
        +1.7105579787824194827314922811552561464 38e-0036L,
        -2.1488377149384067375874890241524646427 38e-0035L,
        -4.2730075413304081440860776955739509433 51e-0035L,
        -1.1875123178301471197422515490901830998 23e-0035L,
        +4.8282717433850775602046156705662770214 63e-0036L,
        +2.8882851313407091276565149486353493118 05e-0035L,
        -4.5052330855803295586842720759044712288 64e-0035L,
        +2.9316308953278386819469845101608839593 32e-0036L,
        +2.6476984851186301144844699759399477913 90e-0035L,
        +3.5893203208453811872540177365316183201 53e-0035L,
        +3.1094095482625904593518474740324158518 43e-0035L,
        +4.0832349108391255310168362697062489227 07e-0035L,
        +2.6170814261859721742789727383114272230 03e-0035L,
        +1.6856388838767364686255988716025670253 29e-0035L,
        +3.3407090070441223621749965155170700740 49e-0035L,
        +4.2724489676747696433358273315135139148 93e-0035L,
        -1.0163370775024549829492877844265875543 12e-0035L,
        -4.1648204724159408772656293740012651810 61e-0035L,
        -1.2667029075294826838554134120285238793 13e-0035L,
        -2.4982955237496767389764157730503099268 89e-0035L,
        -2.2402448880357015285653223020105242166 07e-0035L,
        +2.0726736766330522375123449578397134945 38e-0035L,
        -5.6356205750738490116075473140845111489 18e-0036L,
        +1.2897733987863244444039859257805917099 15e-0035L,
```



```
};
```

```
**********************************************************
    4702 Thu Jun  6 20:38:30 2019
new/usr/src/lib/libm/common/Q/__cosl.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

/* BEGIN CSTYLED */
#endif /* ! codereview */
/*
 * __k_cosl(long double x, long double y)
 * kernel cos function on [-pi/4, pi/4], pi/4 ~ 0.785398164
 * Input x is assumed to be bounded by ~pi/4 in magnitude.
 * Input y is the tail of x.
 *
 * Table look up algorithm
 *      1. by cos(-x) = cos(x), we may replace x by |x|
 *      2. if x < 25/128 = [0x3ffc4000, 0] = 0.15625 , then
 *           if x < 2^-57 (hx < 0x3fc60000 0), return 1.0 with inexact if x !=
 *           z = x*x;
 *           if x <= 1/128 = 2**-7 = 0.0078125
 *              cos(x)=1.0+z*(qq1+z*(qq2+z*(qq3+z*(qq4+z*qq5))))
 *           else
 *              cos(x)=1.0+z*(q1+ ... z*q8)
 *      3. else
 *              ht = (hx + 0x400)&0x7ffff800    (round x to a break point t)
 *              lt = 0
 *              i  = (hy-0x3ffc4000)>>11;       (i<=64)
 *              x' = (x - t)+y                  (|x'| ~<= 2^-7
 *         By
 *              cos(t+x')
 *                = cos(t)cos(x')-sin(t)sin(x')
 *                = cos(t)(1+z*(qq1+z*qq2))-[sin(t)]*x*(1+z*(pp1+z*pp2))
 *                = cos(t) + [cos(t)]*(z*(qq1+z*qq2))-
 *                             [sin(t)]*x*(1+z*(pp1+z*pp2))
 *
 *         Thus,
```



```
 *              let a= _TBL_cos_hi[i], b = _TBL_cos_lo[i], c= _TBL_sin_hi[i],
 *              x = (x-t)+y
 *              z = x*x;
 *              cos(t+x) = a+(b+ (-c*x*(1+z*(pp1+z*pp2))+a*(z*(qq1+z*qq2)))
 */
/* END CSTYLED */
#endif /* ! codereview */

#include "libm.h"

extern const long double _TBL_cosl_hi[], _TBL_cosl_lo[], _TBL_sinl_hi[];
static const long double one = 1.0L;

static const long double
        one     = 1.0L,
/*
 *                    3           11       -122.32
 * |sin(x) - (x+pp1*x +...+ pp5*x  )| <= 2        for |x|<1/64
 */
static const long double
#endif /* ! codereview */
        pp1 = -1.6666666666666666666666666586782940810e-0001L,
        pp2 = +8.3333333333333333333330037236609293175 40e-0003L,
        pp3 = -1.9841269841269840760459034837783378044 70e-0004L,
        pp4 = +2.7557319223619066413197231062109009494 13e-0006L,
        pp5 = -2.5051983985709470190939984691350120576 73e-0008L;

        pp5     = -2.505198398570947019093998469135012057673e-0008L,
/*
 *                  2            16       -117.11
 * |cos(x) - (1+q1*x + ... + q8*x  )| <= 2        for |x|<= 0.15625
 */
static const long double
#endif /* ! codereview */
        q1 = -4.9999999999999999999999999999999756416975e-0001L,
        q2 = +4.1666666666666666666666666640060665772 58e-0002L,
        q3 = -1.3888888888888888888888877700363937169637e-0003L,
        q4 = +2.4801587301587301584944684630318140835 59e-0005L,
        q5 = -2.7557319223985862763228192503560055428 71e-0007L,
        q6 = +2.0876756987674242614419597607298540178 55e-0009L,
        q7 = -1.1470744812396620890724521290107907747 61e-0011L,
        q8 = +4.7777616473996515997306634222635310347 82e-0014L;

        q8      = +4.777761647399651599730663422263531034782e-0014L,
/*
 *                    2           10       -123.84
 * |cos(x) - (1+qq1*x +...+ qq5*x  )| <= 2        for |x|<=1/128
 */
static const long double
#endif /* ! codereview */
        qq1 = -4.9999999999999999999999999999999378373641e-0001L,
        qq2 = +4.1666666666666666666666665478399327703130e-0002L,
        qq3 = -1.3888888888888888880582112306180516134 94e-0003L,
        qq4 = +2.4801587301561053777715856589053031118 66e-0005L,
        qq5 = -2.7557280997625263257364883766951570087 36e-0007L;

#define i0      0

long double
__k_cosl(long double x, long double y)
{
__k_cosl(long double x, long double y) {
        long double a, t, z, w;
        int *pt = (int *)&t, *px = (int *)&x;
        int *pt = (int *) &t, *px = (int *) &x;
        int i, j, hx, ix;
```

```
        t = 1.0L;
        hx = px[i0];
        ix = hx & 0x7fffffff;

#endif /* ! codereview */
        if (ix < 0x3ffc4000) {
                if (ix < 0x3fc60000)
                        if ((i = (int)x) == 0)
                                return (one);

                /* generate inexact */
                        if ((i = (int) x) == 0)
                                return (one);   /* generate inexact */
                z = x * x;

                if (ix < 0x3ff80000) {  /* 0.0078125 */
                        return (one + z * (qq1 + z * (qq2 + z * (qq3 + z *
                            (qq4 + z * qq5)))));
                } else {
                        return (one + z * (q1 + z * (q2 + z * (q3 + z * (q4 +
                            z * (q5 + z * (q6 + z * (q7 + z * q8))))))));
                }
                if (ix < 0x3ff80000)    /* 0.0078125 */
                        return one + z * (qq1 + z * (qq2 + z * (qq3 +
                                z * (qq4 + z * qq5))));
                else
                        return one + z * (q1 + z * (q2 + z * (q3 +
                                z * (q4 + z * (q5 + z * (q6 + z * (q7 +
                                z * q8)))))));
        }

#endif /* ! codereview */
        j = (ix + 0x400) & 0x7ffff800;
        i = (j - 0x3ffc4000) >> 11;
        pt[i0] = j;

#endif /* ! codereview */
        if (hx > 0)
                x = y - (t - x);
        else
                x = (-y) - (t + x);

#endif /* ! codereview */
        a = _TBL_cosl_hi[i];
        z = x * x;
        t = z * (qq1 + z * (qq2 + z * (qq3 + z * (qq4 + z * qq5))));
        w = x * (one + z * (pp1 + z * (pp2 + z * (pp3 + z * (pp4 + z * pp5)))));
        t = _TBL_cosl_lo[i] - (_TBL_sinl_hi[i] * w - a * t);
        return (a + t);
}
```

```
**********************************************************
   14589 Thu Jun  6 20:38:31 2019
new/usr/src/lib/libm/common/Q/__lgammal.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

/*
 * long double __k_lgammal(long double x, int *signgamlp);
 * K.C. Ng, August, 1989.
 *
 * We choose [1.5,2.5] to be the primary interval. Our algorithms
 * are mainly derived from
 *
 *                                zeta(2)-1    2    zeta(3)-1    3
 * lgamma(2+s) = s*(1-euler) + --------- * s  -  --------- * s  + ...
 *                                    2                 3
 *
 *
 * Note 1. Since gamma(1+s)=s*gamma(s), hence
 *              lgamma(1+s) = log(s) + lgamma(s), or
 *              lgamma(s) = lgamma(1+s) - log(s).
 *         When s is really tiny (like roundoff), lgamma(1+s) ~ s(1-enler)
 *         Hence lgamma(s) ~ -log(s) for tiny s
 *
 */

#include "libm.h"
#include "longdouble.h"

static long double neg(long double, int *);
static long double poly(long double, const long double *, int);
static long double polytail(long double);
static long double primary(long double);

static const long double c0 = 0.0L,
```



```
        ch = 0.5L,
        c1 = 1.0L,
        c2 = 2.0L,
        c3 = 3.0L,
        c4 = 4.0L,
        c5 = 5.0L,
        c6 = 6.0L,
        pi = 3.14159265358979323846264338327950288419 71L,
        tiny = 1.0e-40L;
static const long double
c0 =     0.0L,
ch =     0.5L,
c1 =     1.0L,
c2 =     2.0L,
c3 =     3.0L,
c4 =     4.0L,
c5 =     5.0L,
c6 =     6.0L,
pi =     3.14159265358979323846264338327950288419 71L,
tiny =   1.0e-40L;

long double
__k_lgammal(long double x, int *signgamlp)
{
        long double t, y;
__k_lgammal(long double x, int *signgamlp) {
        long double t,y;
        int i;

        /* purge off +-inf, NaN and negative arguments */
        if (!finitel(x))
                return (x * x);

        if (!finitel(x)) return x*x;
        *signgamlp = 1;

        if (signbitl(x))
                return (neg(x, signgamlp));
        if (signbitl(x)) return (neg(x,signgamlp));

        /* for x < 8.0 */
        if (x < 8.0L) {
        if (x<8.0L) {
                y = anintl(x);
                i = (int)y;

                switch (i) {
            i = (int) y;
            switch(i) {
                case 0:

                        if (x < 1.0e-40L)
                                return (-logl(x));
                        else
                                return ((primary(x) - log1pl(x)) - logl(x));

                if (x<1.0e-40L) return -logl(x); else
                return (primary(x)-log1pl(x))-logl(x);
                case 1:
                        return (primary(x - y) - logl(x));
                return primary(x-y)-logl(x);
                case 2:
                        return (primary(x - y));
                return primary(x-y);
                case 3:
                        return (primary(x - y) + logl(x - c1));
```

```
 60                 return primary(x-y)+logl(x-c1);
106                 case 4:
107                         return (primary(x - y) + logl((x - c1) * (x - c2)));
 62                 return primary(x-y)+logl((x-c1)*(x-c2));
108                 case 5:
109                         return (primary(x - y) + logl((x - c1) * (x - c2) * (x -
110                             c3)));
 64                 return primary(x-y)+logl((x-c1)*(x-c2)*(x-c3));
111                 case 6:
112                         return (primary(x - y) + logl((x - c1) * (x - c2) * (x -
113                             c3) * (x - c4)));
 66                 return primary(x-y)+logl((x-c1)*(x-c2)*(x-c3)*(x-c4));
114                 case 7:
115                         return (primary(x - y) + logl((x - c1) * (x - c2) * (x -
116                             c3) * (x - c4) * (x - c5)));
 68                 return primary(x-y)+logl((x-c1)*(x-c2)*(x-c3)*(x-c4)*(x-c5));
117                 case 8:
118                         return (primary(x - y) + logl((x - c1) * (x - c2) * (x -
119                             c3) * (x - c4) * (x - c5) * (x - c6)));
 70                 return primary(x-y)+
 71                         logl((x-c1)*(x-c2)*(x-c3)*(x-c4)*(x-c5)*(x-c6));
120                 }
121         }

123         /* 8.0 <= x < 1.0e40 */
124         if (x < 1.0e40L) {
125                 t = logl(x);
126                 return (x * (t - c1) - (ch * t - polytail(c1 / x)));
 78             return x*(t-c1)-(ch*t-polytail(c1/x));
127         }

129         /* 1.0e40 <= x <= inf */
130         return (x * (logl(x) - c1));
 82         return x*(logl(x)-c1);
131 }
```

_____*unchanged_portion_omitted_*

```
314 static long double
315 primary(long double s)                  /* assume |s|<=0.5 */
316 {
267 primary(long double s) {        /* assume |s|<=0.5 */
317         int i;

319         i = (int)(8.0L * (s + 0.5L));

321         switch (i) {
322         case 0:
323                 return (ch * s + s * poly(s, an4, 21));
324         case 1:
325                 return (ch * s + s * poly(s, an3, 20));
326         case 2:
327                 return (ch * s + s * poly(s, an2, 20));
328         case 3:
329                 return (ch * s + s * poly(s, an1, 20));
330         case 4:
331                 return (ch * s + s * poly(s, ap1, 19));
332         case 5:
333                 return (ch * s + s * poly(s, ap2, 19));
334         case 6:
335                 return (ch * s + s * poly(s, ap3, 19));
336         case 7:
337                 return (ch * s + s * poly(s, ap4, 19));
270         i = (int) (8.0L * (s + 0.5L));
271         switch(i) {
272         case 0: return ch*s+s*poly(s,an4,21);
273         case 1: return ch*s+s*poly(s,an3,20);
```



```
274         case 2: return ch*s+s*poly(s,an2,20);
275         case 3: return ch*s+s*poly(s,an1,20);
276         case 4: return ch*s+s*poly(s,ap1,19);
277         case 5: return ch*s+s*poly(s,ap2,19);
278         case 6: return ch*s+s*poly(s,ap3,19);
279         case 7: return ch*s+s*poly(s,ap4,19);
338         }

340 #endif /* ! codereview */
341         /* NOTREACHED */
342         return (0.0L);
281     return 0.0L;
343 }

345 static long double
346 poly(long double s, const long double *p, int n)
347 {
285 poly(long double s, const long double *p, int n) {
348         long double y;
349         int i;

351         y = p[n - 1];

353         for (i = n - 2; i >= 0; i--)
354                 y = p[i] + s * y;

356         return (y);
288         y = p[n-1];
289         for (i=n-2;i>=0;i--) y = p[i]+s*y;
290         return y;
357 }
```

_____*unchanged_portion_omitted_*

```
381 static long double
382 polytail(long double s)
383 {
384         long double t, z;
316 polytail(long double s) {
317         long double t,z;
385         int i;

387         z = s * s;
319         z = s*s;
388         t = pt[18];

390         for (i = 17; i >= 1; i--)
391                 t = pt[i] + z * t;

393         return (pt[0] + s * t);
321         for (i=17;i>=1;i--) t = pt[i]+z*t;
322         return pt[0]+s*t;
394 }

396 static long double
397 neg(long double z, int *signgamlp)
398 {
399         long double t, p;
326 neg(long double z, int *signgamlp) {
327         long double t,p;

401         /* BEGIN CSTYLED */
402 #endif /* ! codereview */
403         /*
404          * written by K.C. Ng,  Feb 2, 1989.
405          *
406          * Since
```

```
 407          *              -z*G(-z)*G(z) = pi/sin(pi*z),
 408          * we have
 409          *      G(-z) = -pi/(sin(pi*z)*G(z)*z)
 410          *            =  pi/(sin(pi*(-z))*G(z)*z)
 411          * Algorithm
 412          *              z = |z|
 413          *              t = sinpi(z); ...note that when z>2**112, z is an int
 414          *              and hence t=0.
 415          *
 416          *              if (t == 0.0) return 1.0/0.0;
 417          *              if (t< 0.0) *signgamlp = -1; else t= -t;
 418          *              if (z<1.0e-40)  ...tiny z
 419          *                  return -log(z);
 420          *              else
 421          *                  return log(pi/(t*z))-lgamma(z);
 422          *
 423          */
 424         /* END CSTYLED */
 425 #endif /* ! codereview */

 427         t = sinpil(z);                  /* t := sin(pi*z) */

 429 #endif /* ! codereview */
 430         if (t == c0)                    /* return   1.0/0.0 =  +INF */
 431                 return (c1 / c0);
 329             return c1/c0;

 433         z = -z;

 435         if (z <= tiny)
 332         if (z<=tiny)
 436                 p = -logl(z);
 437         else
 438                 p = logl(pi / (fabsl(t) * z)) - __k_lgammal(z, signgamlp);

 440         if (t < c0)
 441                 *signgamlp = -1;

 443         return (p);
 335             p = logl(pi/(fabsl(t)*z))-__k_lgammal(z,signgamlp);
 336         if (t<c0) *signgamlp = -1;
 337         return p;
 444 }
```

_____*unchanged_portion_omitted_*

```
**********************************************************
    1196 Thu Jun  6 20:38:31 2019
new/usr/src/lib/libm/common/Q/__poly_libmq.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#include "libm.h"

long double
__poly_libmq(long double x, int n, const long double p[])
{
__poly_libmq(long double x, int n, const long double p[]) {
        long double t;
        int i;

        t = p[n - 1];

#endif /* ! codereview */
        for (i = n - 2; i >= 0; i--)
                t = p[i] + x * t;

#endif /* ! codereview */
        return (t);
}
```

```
**********************************************************
    2203 Thu Jun  6 20:38:32 2019
new/usr/src/lib/libm/common/Q/__rem_pio2l.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

/*
 * __rem_pio2l(x,y)
 *
 * return the remainder of x rem pi/2 in y[0]+y[1] by calling __rem_pio2m
 */

#ifndef FDLIBM_BASED
#include "libm.h"

#endif /* ! codereview */
extern int __rem_pio2m(double *, double *, int, int, int, const int *);
#else /* FDLIBM_BASED */
#include "fdlibm.h"
#define __rem_pio2m     __kernel_rem_pio2
#endif /* FDLIBM_BASED */

#include "longdouble.h"

extern const int _TBL_ipio2l_inf[];
static const long double two24l = 16777216.0L,

static const long double
        two24l = 16777216.0L,
        pio4 = 0.785398163397448309615660845819875721049 5L;

int
__rem_pio2l(long double x, long double *y)
{
        long double z, w;
        double t[5], v[5];
```



```
        int e0, i, nx, n, sign;
        const int *ipio2;

        sign = signbitl(x);
        z = fabsl(x);

#endif /* ! codereview */
        if (z <= pio4) {
                y[0] = x;
                y[1] = 0;
                return (0);
        }

#endif /* ! codereview */
        e0 = ilogbl(z) - 23;
        z = scalbnl(z, -e0);

#endif /* ! codereview */
        for (i = 0; i < 5; i++) {
                t[i] = (double)((int)(z));
                z = (z - (long double)t[i]) * two24l;
        }

#endif /* ! codereview */
        nx = 5;

#endif /* ! codereview */
        while (t[nx - 1] == 0.0)
                nx--;                   /* skip zero term */

#endif /* ! codereview */
        ipio2 = _TBL_ipio2l_inf;
        n = __rem_pio2m(t, v, e0, nx, 3, (const int *)ipio2);
        z = (long double)v[2] + (long double)v[1];
        w = (long double)v[0];
        y[0] = z + w;
        y[1] = z - (y[0] - w);

#endif /* ! codereview */
        if (sign != 0) {
                y[0] = -y[0];
                y[1] = -y[1];
                return (-n);
        }

#endif /* ! codereview */
        return (n);
}
```

```
**********************************************************
    4909 Thu Jun  6 20:38:32 2019
new/usr/src/lib/libm/common/Q/__sincosl.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

/*
 * long double __k_sincos(long double x, long double y, long double *c);
 * kernel sincosl function on [-pi/4, pi/4], pi/4 ~ 0.785398164
 * Input x is assumed to be bounded by ~pi/4 in magnitude.
 * Input y is the tail of x.
 * return sinl(x) with *c = cosl(x)
 *
 * Table look up algorithm
 *      see __k_sinl and __k_cosl
 */

#include "libm.h"

extern const long double _TBL_sinl_hi[], _TBL_sinl_lo[], _TBL_cosl_hi[],
        _TBL_cosl_lo[];
static const long double one = 1.0L;

extern const long double _TBL_sinl_hi[], _TBL_sinl_lo[],
        _TBL_cosl_hi[], _TBL_cosl_lo[];
static const long double
        one     = 1.0L,
/*
 *                    3           11       -122.32
 * |sin(x) - (x+pp1*x +...+ pp5*x  )| <= 2        for |x|<1/64
 */
static const long double
#endif /* ! codereview */
        pp1 = -1.6666666666666666666666666586782940810e-0001L,
        pp2 = +8.3333333333333333333300372366092931754 0e-0003L,
        pp3 = -1.9841269841269840760459034837783378 04470e-0004L,
```



```
        pp4 = +2.7557319223619066413197231062109009494 13e-0006L,
        pp5 = -2.5051983985709470190939984691350120576 73e-0008L;

        pp5     = -2.5051983985709470190939984691350120576 73e-0008L,
/*
 * |(sin(x) - (x+p1*x^3+...+p8*x^17))|
 * |--------------------------------- | <= 2^-116.17 for |x|<0.1953125
 * |                  x               |
 */
static const long double
#endif /* ! codereview */
        p1 = -1.6666666666666666666666666666211262297e-0001L,
        p2 = +8.3333333333333333333333333014978769085 41e-0003L,
        p3 = -1.9841269841269841269840413028811806219 22e-0004L,
        p4 = +2.7557319223985890641005873513072696210 93e-0006L,
        p5 = -2.5052108385441631293789069537655959393 873e-0008L,
        p6 = +1.6059043836432443750509982437785340742 73e-0010L,
        p7 = -7.6471627228006855169014561142708246226 99e-0013L,
        p8 = +2.8100464286619029617254288410688444626 03e-0015L;

        p8      = +2.8100464286619029617254288410688444626 03e-0015L,
/*
 *                    2           10       -123.84
 * |cos(x) - (1+qq1*x +...+ qq5*x  )| <= 2        for |x|<=1/128
 */
static const long double
#endif /* ! codereview */
        qq1 = -4.9999999999999999999999999999999378373641e-0001L,
        qq2 = +4.1666666666666666666666547839932770313 0e-0002L,
        qq3 = -1.3888888888888888880582112306180516134 94e-0003L,
        qq4 = +2.4801587301561053777715856589053031118 66e-0005L,
        qq5 = -2.7557280997625263257364883766957157008 736e-0007L;

        qq5     = -2.7557280997625263257364883766957157008 736e-0007L,
/*
 *                   2            16       -117.11
 * |cos(x) - (1+q1*x + ... + q8*x  )| <= 2        for |x|<= 0.15625
 */
static const long double
#endif /* ! codereview */
        q1 = -4.9999999999999999999999999999999756416975e-0001L,
        q2 = +4.1666666666666666666666666400606657725 8e-0002L,
        q3 = -1.3888888888888888888888877700363937169637e-0003L,
        q4 = +2.4801587301587301584944684630318140835 59e-0005L,
        q5 = -2.7557319223985862763228192503560055428 71e-0007L,
        q6 = +2.0876756987674242614419597607298540178 55e-0009L,
        q7 = -1.1470744812396620890724521290107907747 61e-0011L,
        q8 = +4.7777616473996515997306634222635310347 82e-0014L;

#define i0      0

long double
__k_sincosl(long double x, long double y, long double *c)
{
__k_sincosl(long double x, long double y, long double *c) {
        long double a1, a2, t, t1, t2, z, w;
        int *pt = (int *)&t, *px = (int *)&x;
        int *pt = (int *) &t, *px = (int *) &x;
        int i, j, hx, ix;

        t = 1.0L;
        hx = px[i0];
        ix = hx & 0x7fffffff;

#endif /* ! codereview */
        if (ix < 0x3ffc4000) {
```

```
                if (ix < 0x3fc60000) {
                        if (((int)x) == 0) {
 57             if (ix < 0x3fc60000)
 58                     if (((int) x) == 0) {
                                *c = one;
                                return (x);
                        }               /* generate inexact */
                }

#endif /* ! codereview */
                z = x * x;

                if (ix < 0x3ff80000) {
                        *c = one + z * (qq1 + z * (qq2 + z * (qq3 + z * (qq4 +
                            z * qq5))));
                        t = z * (p1 + z * (p2 + z * (p3 + z * (p4 + z * (p5 +
                            z * p6)))));
 62                     *c = one + z * (qq1 + z * (qq2 + z * (qq3 +
 63                             z * (qq4 + z * qq5))));
 64                     t = z * (p1 + z * (p2 + z * (p3 + z * (p4 +
 65                             z * (p5 + z * p6)))));
                } else {
                        *c = one + z * (q1 + z * (q2 + z * (q3 + z * (q4 + z *
                            (q5 + z * (q6 + z * (q7 + z * q8)))))));
 67                     *c = one + z * (q1 + z * (q2 + z * (q3 + z * (q4 +
 68                             z * (q5 + z * (q6 + z * (q7 + z * q8)))))));
                        t = z * (p1 + z * (p2 + z * (p3 + z * (p4 + z * (p5 +
                            z * (p6 + z * (p7 + z * p8)))))));
                }

                t = y + x * t;
                return (x + t);
        }

#endif /* ! codereview */
        j = (ix + 0x400) & 0x7ffff800;
        i = (j - 0x3ffc4000) >> 11;
        pt[i0] = j;

#endif /* ! codereview */
        if (hx > 0)
                x = y - (t - x);
        else
                x = (-y) - (t + x);

#endif /* ! codereview */
        a1 = _TBL_sinl_hi[i];
        z = x * x;
        t = z * (qq1 + z * (qq2 + z * (qq3 + z * (qq4 + z * qq5))));
        w = x * (one + z * (pp1 + z * (pp2 + z * (pp3 + z * (pp4 + z * pp5)))));
        a2 = _TBL_cosl_hi[i];
        t2 = _TBL_cosl_lo[i] - (a1 * w - a2 * t);
        *c = a2 + t2;
        t1 = a2 * w + a1 * t;
        t1 += _TBL_sinl_lo[i];

#endif /* ! codereview */
        if (hx < 0)
                return (-a1 - t1);
        else
                return (a1 + t1);
}
```

```
**********************************************************
    4808 Thu Jun  6 20:38:33 2019
new/usr/src/lib/libm/common/Q/__sinl.c
11210 libm should be cstyle(i0NBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

/*
 * long double __k_sinl(long double x, long double y);
 * kernel sin function on [-pi/4, pi/4], pi/4 ~ 0.785398164
 * Input x is assumed to be bounded by ~pi/4 in magnitude.
 * Input y is the tail of x.
 *
 * Table look up algorithm
 *      1. by sin(-x) = -sin(x), need only to consider positive x
 *      2. if x < 25/128 = [0x3ffc9000,0,0,0] = 0.1953125 , then
 *           if x < 2^-57 (hx < 0x3fc60000,0,0,0), return x (inexact if x !=  0)
 *           z = x*x;
 *           if x <= 1/64 = 2**-6
 *              sin(x) = x + (y+(x*z)*(p1 + z*p2))
 *           else
 *              sin(x) = x + (y+(x*z)*(p1 + z*(p2 + z*(p3 + z*p4))))
 *      3. else
 *              ht = (hx + 0x400)&0x7ffff800    (round x to a break point t)
 *              lt = 0
 *              i  = (hy-0x3ffc4000)>>11;       (i<=64)
 *              x' = (x - t)+y                  (|x'| ~<= 2^-7
 *         By
 *              sin(t+x')
 *                = sin(t)cos(x')+cos(t)sin(x')
 *                = sin(t)(1+z*(qq1+z*qq2))+[cos(t)]*x*(1+z*(pp1+z*pp2))
 *                = sin(t) + [sin(t)]*(z*(qq1+z*qq2))+
 *                               [cos(t)]*x*(1+z*(pp1+z*pp2))
 *
 *         Thus,
 *              let a= _TBL_sin_hi[i], b = _TBL_sin_lo[i], c= _TBL_cos_hi[i],
 *              x = (x-t)+y
```



```
 *              z = x*x;
 *              sin(t+x) = a+(b+ ((c*x)*(1+z*(pp1+z*pp2))+a*(z*(qq1+z*qq2)))
 */

#include "libm.h"

extern const long double _TBL_sinl_hi[], _TBL_sinl_lo[], _TBL_cosl_hi[];
static const long double one = 1.0L;

static const long double
one     = 1.0L,
/*
 *                   3           11        -122.32
 * |sin(x) - (x+pp1*x +...+ pp5*x  )| <= 2        for |x|<1/64
 */
static const long double
#endif /* ! codereview */
        pp1 = -1.6666666666666666666666666666586782940810e-0001L,
        pp2 = +8.3333333333333333333330037236609293175 40e-0003L,
        pp3 = -1.9841269841269840760459034837783378044 70e-0004L,
        pp4 = +2.7557319223619066413197231062109009494 13e-0006L,
        pp5 = -2.5051983985709470190939984691350120576 73e-0008L;

        pp5     = -2.505198398570947019093998469135012057673e-0008L,
/*
 * |(sin(x) - (x+p1*x^3+...+p8*x^17)|
 * |------------------------------- | <= 2^-116.17 for |x|<0.1953125
 * |                 x              |
 */
static const long double
#endif /* ! codereview */
        p1 = -1.6666666666666666666666666666666211262297e-0001L,
        p2 = +8.3333333333333333333333333301497876908541e-0003L,
        p3 = -1.9841269841269841269840413028811806219 22e-0004L,
        p4 = +2.7557319223985890641005873513072696210 93e-0006L,
        p5 = -2.5052108385441631293789069537655953938 73e-0008L,
        p6 = +1.6059043836432443750509982437785340742 73e-0010L,
        p7 = -7.6471627228006855169014561142708246226 99e-0013L,
        p8 = +2.8100464286619029617254288410688444626 03e-0015L;

        p8      = +2.810046428661902961725428841068844462603e-0015L,
/*
 *                   2           10        -123.84
 * |cos(x) - (1+qq1*x +...+ qq5*x  )| <= 2        for |x|<=1/128
 */
static const long double
#endif /* ! codereview */
        qq1 = -4.9999999999999999999999999999999378373641e-0001L,
        qq2 = +4.1666666666666666666666654783993277031 30e-0002L,
        qq3 = -1.3888888888888888880582112306180516134 94e-0003L,
        qq4 = +2.4801587301561053777715856589053031118 66e-0005L,
        qq5 = -2.7557280997625263257364883766951570087 36e-0007L;

#define i0      0

long double
__k_sinl(long double x, long double y)
{
__k_sinl(long double x, long double y) {
        long double a, t, z, w;
        int *pt = (int *)&t, *px = (int *)&x;
        int *pt = (int *) &t, *px = (int *) &x;
        int i, j, hx, ix;

        t = 1.0L;
        hx = px[i0];
```

```
        ix = hx & 0x7fffffff;

#endif /* ! codereview */
        if (ix < 0x3ffc9000) {
                *(3 - i0 + (int *)&t) = -1;     /* one-ulp */
                *(2 + (int *)&t) = -1;          /* one-ulp */
                *(1 + (int *)&t) = -1;          /* one-ulp */
                *(i0 + (int *)&t) -= 1;         /* one-ulp */

                *(3 - i0 + (int *) &t) = -1;    /* one-ulp */
                *(2 + (int *) &t) = -1; /* one-ulp */
                *(1 + (int *) &t) = -1; /* one-ulp */
                *(i0 + (int *) &t) -= 1;        /* one-ulp */
                if (ix < 0x3fc60000)
                        if (((int)(x * t)) < 1)
                                return (x);

                /* inexact and underflow */
                        if (((int) (x * t)) < 1)
                                return (x);     /* inexact and underflow */
                z = x * x;
                t = z * (p1 + z * (p2 + z * (p3 + z * (p4 + z * (p5 + z * (p6 +
                    z * (p7 + z * p8)))))));
                t = z * (p1 + z * (p2 + z * (p3 + z * (p4 + z * (p5 +
                        z * (p6 + z * (p7 + z * p8)))))));
                t = y + x * t;
                return (x + t);
        }

#endif /* ! codereview */
        j = (ix + 0x400) & 0x7ffff800;
        i = (j - 0x3ffc4000) >> 11;
        pt[i0] = j;

#endif /* ! codereview */
        if (hx > 0)
                x = y - (t - x);
        else
                x = (-y) - (t + x);

#endif /* ! codereview */
        a = _TBL_sinl_hi[i];
        z = x * x;
        t = z * (qq1 + z * (qq2 + z * (qq3 + z * (qq4 + z * qq5))));
        w = x * (one + z * (pp1 + z * (pp2 + z * (pp3 + z * (pp4 + z * pp5)))));
        t = _TBL_cosl_hi[i] * w + a * t;
        t += _TBL_sinl_lo[i];

#endif /* ! codereview */
        if (hx < 0)
                return (-a - t);
        else
                return (a + t);
}
```

```
**********************************************************
    5586 Thu Jun  6 20:38:33 2019
new/usr/src/lib/libm/common/Q/__tanl.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

/*
 * long double __k_tanl(long double x; long double y, int k);
 * kernel tan/cotan function on [-pi/4, pi/4], pi/4 ~ 0.785398164
 * Input x is assumed to be bounded by ~pi/4 in magnitude.
 * Input y is the tail of x.
 * Input k indicate -- tan if k=0; else -1/tan
 *
 * Table look up algorithm
 *      1. by tan(-x) = -tan(x), need only to consider positive x
 *      2. if x < 5/32 = [0x3ffc4000, 0] = 0.15625 , then
 *           if x < 2^-57 (hx < 0x3fc40000 0), set w=x with inexact if x !=  0
 *           else
 *              z = x*x;
 *              w = x + (y+(x*z)*(t1+z*(t2+z*(t3+z*(t4+z*(t5+z*t6))))))
 *         return (k == 0)? w: 1/w;
 *      3. else
 *              ht = (hx + 0x400)&0x7ffff800    (round x to a break point t)
 *              lt = 0
 *              i  = (hy-0x3ffc4000)>>11;       (i<=64)
 *              x' = (x - t)+y                  (|x'| ~<= 2^-7)
 *         By
 *              tan(t+x')
 *                = (tan(t)+tan(x'))/(1-tan(x')tan(t))
 *         We have
 *                            sin(x')+tan(t)*(tan(t)*sin(x'))
 *                = tan(t) + ------------------------------    for k=0
 *                               cos(x') - tan(t)*sin(x')
 *
 *                            cos(x') - tan(t)*sin(x')
 *                = - -------------------------------------    for k=1
```



```
 *                      tan(t) + tan(t)*(cos(x')-1) + sin(x')
 *
 *
 *         where        tan(t) is from the table,
 *                      sin(x') = x + pp1*x^3 + ...+ pp5*x^11
 *                      cos(x') = 1 + qq1*x^2 + ...+ qq5*x^10
 */

#include "libm.h"

extern const long double _TBL_tanl_hi[], _TBL_tanl_lo[];
static const long double one = 1.0L;

static const long double
        one     = 1.0L,
/*
 *                    3           11       -122.32
 * |sin(x) - (x+pp1*x +...+ pp5*x  )| <= 2        for |x|<1/64
 */
static const long double
#endif /* ! codereview */
        pp1 = -1.6666666666666666666666666586782940810e-0001L,
        pp2 = +8.3333333333333333333330037236609293175 40e-0003L,
        pp3 = -1.9841269841269840760459034837783378044 70e-0004L,
        pp4 = +2.7557319223619066413197231062109009494 13e-0006L,
        pp5 = -2.5051983985709470190939984691350120576 73e-0008L;

        pp5     = -2.505198398570947019093998469135012057673e-0008L,
/*
 *                    2           10        -123.84
 * |cos(x) - (1+qq1*x +...+ qq5*x  )| <= 2        for |x|<=1/128
 */
static const long double
#endif /* ! codereview */
        qq1 = -4.9999999999999999999999999999999378373641e-0001L,
        qq2 = +4.1666666666666666666666654783993277031 30e-0002L,
        qq3 = -1.3888888888888888888058211230618051613494e-0003L,
        qq4 = +2.4801587301561053777715856589053031118 66e-0005L,
        qq5 = -2.7557280997625263257364883766951570087 36e-0007L;

        qq5     = -2.755728099762526325736488376695157008736e-0007L,
/*
 * |tan(x) - (x+t1*x^3+...+t6*x^13)|
 * |------------------------------ | <= 2^-59.73 for |x|<0.15625
 * |               x               |
 */
static const long double
#endif /* ! codereview */
        t1 = +3.3333333333333333333333333333333423342490e-0001L,
        t2 = +1.3333333333333333333333333333093838744537e-0001L,
        t3 = +5.3968253968253968253968279063186826622 50e-0002L,
        t4 = +2.1869488536155202821855769769944184869 11e-0002L,
        t5 = +8.8632355299021965733545545199911529362 46e-0003L,
        t6 = +3.5921280365724800646521914275439948787 90e-0003L,
        t7 = +1.4558343870514552578568338075819013054 74e-0003L,
        t8 = +5.9002744093185998578299832562017255874 77e-0004L,
        t9 = +2.3912911521172651815011169619011223629 37e-0004L,
        t10 = +9.6915331693827297423940241731949818823 75e-0005L,
        t11 = +3.9279947331864156032281781842257808599 51e-0005L,
        t12 = +1.5883000188483238242276400648833341012 88e-0005L,
        t13 = +6.9162712233968083111662022851317222317 23e-0006L;

#define i0      0

long double
__k_tanl(long double x, long double y, int k)
```

```
124 {
 42 __k_tanl(long double x, long double y, int k) {
125         long double a, t, z, w = 0, s, c;
126         int *pt = (int *)&t, *px = (int *)&x;
 44         int *pt = (int *) &t, *px = (int *) &x;
127         int i, j, hx, ix;

129         t = 1.0L;
130         hx = px[i0];
131         ix = hx & 0x7fffffff;

133 #endif /* ! codereview */
134         if (ix < 0x3ffc4000) {
135                 *(3 - i0 + (int *)&t) = 1;      /* make t = one+ulp */

 50                 *(3 - i0 + (int *) &t) = 1;     /* make t = one+ulp */
137                 if (ix < 0x3fc60000) {
138                         if (((int)(x * t)) < 1) /* generate inexact */
 52                         if (((int) (x * t)) < 1)        /* generate inexact */
139                                 w = x; /* generate underflow if subnormal */
140                 } else {
141                         z = x * x;

143                         if (ix < 0x3ff30000) {  /* 2**-12 */
 56                         if (ix < 0x3ff30000)    /* 2**-12 */
144                                 t = z * (t1 + z * (t2 + z * (t3 + z * t4)));
145                         } else {
146                                 t = z * (t1 + z * (t2 + z * (t3 + z * (t4 + z *
147                                     (t5 + z * (t6 + z * (t7 + z * (t8 + z *
148                                     (t9 + z * (t10 + z * (t11 + z * (t12 + z *
149                                     t13)))))))))))));
150                         }

 58                         else
 59                                 t = z * (t1 + z * (t2 + z * (t3 + z * (t4 +
 60                                         z * (t5 + z * (t6 + z * (t7 + z * (t8 +
 61                                         z * (t9 + z * (t10 + z * (t11 +
 62                                         z * (t12 + z * t13)))))))))))));
152                         t = y + x * t;
153                         w = x + t;
154                 }

156 #endif /* ! codereview */
157                 return (k == 0 ? w : -one / w);
158         }

160 #endif /* ! codereview */
161         j = (ix + 0x400) & 0x7ffff800;
162         i = (j - 0x3ffc4000) >> 11;
163         pt[i0] = j;

165 #endif /* ! codereview */
166         if (hx > 0)
167                 x = y - (t - x);
168         else
169                 x = (-y) - (t + x);

171 #endif /* ! codereview */
172         a = _TBL_tanl_hi[i];
173         z = x * x;
174         /* cos(x)-1 */
175         t = z * (qq1 + z * (qq2 + z * (qq3 + z * (qq4 + z * qq5))));
176         /* sin(x) */
177         s = x * (one + z * (pp1 + z * (pp2 + z * (pp3 + z * (pp4 + z * pp5)))));

179 #endif /* ! codereview */
```



```
180         if (k == 0) {
181                 w = a * s;
182                 t = _TBL_tanl_lo[i] + (s + a * w) / (one - (w - t));
183                 return (hx < 0 ? -a - t : a + t);
184         } else {
185                 w = s + a * t;
186                 c = w + _TBL_tanl_lo[i];
187                 z = one - (a * s - t);
188                 return (hx >= 0 ? z / (-a - c) : z / (a + c));
189         }
190 }
```

```
**********************************************************
    1488 Thu Jun  6 20:38:34 2019
new/usr/src/lib/libm/common/Q/acoshl.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
   1 /*
   2  * CDDL HEADER START
   3  *
   4  * The contents of this file are subject to the terms of the
   5  * Common Development and Distribution License (the "License").
   6  * You may not use this file except in compliance with the License.
   7  *
   8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
   9  * or http://www.opensolaris.org/os/licensing.
  10  * See the License for the specific language governing permissions
  11  * and limitations under the License.
  12  *
  13  * When distributing Covered Code, include this CDDL HEADER in each
  14  * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  15  * If applicable, add the following below this CDDL HEADER, with the
  16  * fields enclosed by brackets "[]" replaced with your own identifying
  17  * information: Portions Copyright [yyyy] [name of copyright owner]
  18  *
  19  * CDDL HEADER END
  20  */

  22 /*
  23  * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
  24  */

  26 #endif /* ! codereview */
  27 /*
  28  * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
  29  * Use is subject to license terms.
  30  */

  32 #pragma weak __acoshl = acoshl

  34 #include "libm.h"

  36 static const long double zero = 0.0L,
  25 static const long double
  26         zero    = 0.0L,
  37         ln2 = 6.931471805599453094172321214581765680755e-0001L,
  38         one = 1.0L,
  39         big = 1.e+20L;

  41 long double
  42 acoshl(long double x)
  43 {
  32 acoshl(long double x) {
  44         long double t;

  46         if (isnanl(x)) {
  35         if (isnanl(x))
  47                 return (x + x);
  48         } else if (x > big) {
  37         else if (x > big)
  49                 return (logl(x) + ln2);
  50         } else if (x > one) {
  39         else if (x > one) {
  51                 t = sqrtl(x - one);
  52                 return (log1pl(t * (t + sqrtl(x + one))));
  53         } else if (x == one) {
  42         } else if (x == one)
  54                 return (zero);
```



```
  55         } else {
  44         else
  56                 return ((x - x) / (x - x));
  57         }
  58 #endif /* ! codereview */
  59 }
```

```
**********************************************************
    1828 Thu Jun  6 20:38:34 2019
new/usr/src/lib/libm/common/Q/acosl.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

/*
 * arccosin function
 *                                 _______
 *                                / 1 - x
 *      acos(x) = 2*atan2(  / -------- , 1 )
 *                        \/    1 + x
 *
 *                                 _______
 *                                / 1 - x
 *              = 2*atan (  / -------- ) for non-exceptional x.
 *                        \/    1 + x
 *
 * Special cases:
 *      if x is NaN, return x itself;
 *      if |x|>1, return NaN with invalid signal.
 */

#pragma weak __acosl = acosl

#include "libm.h"

static const long double zero = 0.0L, one = 1.0L;

long double
acosl(long double x)
{
acosl(long double x) {
        if (isnanl(x))
                return (x + x);
        else if (fabsl(x) < one)
```



```
                x = atanl(sqrtl((one - x) / (one + x)));
        else if (x == -one)
                x = atan2l(one, zero);  /* x <- PI */
        else if (x == one)
                x = zero;
        else                            /* |x| > 1  create invalid signal */
        else {          /* |x| > 1  create invalid signal */
                return (zero / zero);

        }
        return (x + x);
}
______unchanged_portion_omitted_
```

```
**********************************************************
    1621 Thu Jun  6 20:38:35 2019
new/usr/src/lib/libm/common/Q/asinhl.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __asinhl = asinhl

#include "libm.h"

static const long double
        ln2 = 6.931471805599453094172321214581765680755e-0001L,
        one = 1.0L,
        big = 1.0e+20L,
        tiny = 1.0e-20L;

long double
asinhl(long double x)
{
asinhl(long double x) {
        long double t, w;

#endif /* ! codereview */
#ifndef lint
        volatile long double dummy;
#endif

        w = fabsl(x);

#endif /* ! codereview */
        if (isnanl(x))
                return (x + x);         /* x is NaN */

#endif /* ! codereview */
        if (w < tiny) {
#ifndef lint
```



```
                dummy = x + big;        /* inexact if x != 0 */
#endif
                return (x);             /* tiny x */
        } else if (w < big) {
                t = one / w;
                return (copysignl(log1pl(w + w / (t + sqrtl(one + t * t))), x));
        } else {
        } else
                return (copysignl(logl(w) + ln2, x));
        }
#endif /* ! codereview */
}
```

```
**********************************************************
    2049 Thu Jun  6 20:38:35 2019
new/usr/src/lib/libm/common/Q/asinl.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __asinl = asinl

/*
 *      asinl(x) = atan2l(x,sqrt(1-x*x));
 *
 * For better accuracy, 1-x*x is computed as follows
 *      1-x*x                     if x <  0.5,
 *      2*(1-|x|)-(1-|x|)*(1-|x|) if x >= 0.5.
 *
 * Special cases:
 *      if x is NaN, return x itself;
 *      if |x|>1, return NaN with invalid signal.
 */

#include "libm.h"

static const long double zero = 0.0L, small = 1.0e-20L, half = 0.5L, one = 1.0L;

#endif /* ! codereview */
#ifndef lint
static const long double big = 1.0e+20L;
#endif

long double
asinl(long double x)
{
asinl(long double x) {
        long double t, w;

#endif /* ! codereview */
```



```
#ifndef lint
        volatile long double dummy;
#endif

        w = fabsl(x);

        if (isnanl(x)) {
        if (isnanl(x))
                return (x + x);
        } else if (w <= half) {
        else if (w <= half) {
                if (w < small) {
#ifndef lint
                        dummy = w + big;
                        /* inexact if w != 0 */
#endif
                        return (x);
                } else {
                } else
                        return (atanl(x / sqrtl(one - x * x)));
                }
#endif /* ! codereview */
        } else if (w < one) {
                t = one - w;
                w = t + t;
                return (atanl(x / sqrtl(w - t * t)));
        } else if (w == one) {
        } else if (w == one)
                return (atan2l(x, zero));       /* asin(+-1) =  +- PI/2 */
        } else {
        else
                return (zero / zero);           /* |x| > 1: invalid */
        }
#endif /* ! codereview */
}
```

```
**********************************************************
    4223 Thu Jun  6 20:38:36 2019
new/usr/src/lib/libm/common/Q/atan2l.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

/*
 * atan2l(y,x)
 *
 * Method :
 *      1. Reduce y to positive by atan2(y,x)=-atan2(-y,x).
 *      2. Reduce x to positive by (if x and y are unexceptional):
 *              ARG (x+iy) = arctan(y/x)           ... if x > 0,
 *              ARG (x+iy) = pi - arctan[y/(-x)]   ... if x < 0,
 *
 * Special cases:
 *
 *      ATAN2((anything), NaN ) is NaN;
 *      ATAN2(NAN , (anything) ) is NaN;
 *      ATAN2(+-0, +(anything but NaN)) is +-0  ;
 *      ATAN2(+-0, -(anything but NaN)) is +-PI ;
 *      ATAN2(+-(anything but 0 and NaN), 0) is +-PI/2;
 *      ATAN2(+-(anything but INF and NaN), +INF) is +-0 ;
 *      ATAN2(+-(anything but INF and NaN), -INF) is +-PI;
 *      ATAN2(+-INF,+INF ) is +-PI/4 ;
 *      ATAN2(+-INF,-INF ) is +-3PI/4;
 *      ATAN2(+-INF, (anything but,0,NaN, and INF)) is +-PI/2;
 *
 * Constants:
 * The hexadecimal values are the intended ones for the following constants.
 * The decimal values may be used, provided that the compiler will convert
 * from decimal to binary accurately enough to produce the hexadecimal values
 * shown.
 */

#pragma weak __atan2l = atan2l
```



```
#include "libm.h"
#include "longdouble.h"

static const long double zero = 0.0L,
static const long double
        zero    =  0.0L,
        tiny = 1.0e-40L,
        one = 1.0L,
        half = 0.5L,
        PI3o4 = 2.356194490192344928846982537459627163148L,
        PIo4 = 0.785398163397448309615660845819875721049L,
        PIo2 = 1.570796326794896619231321691639751442099L,
        PI = 3.141592653589793238462643383279502884197L,
        PI_lo = 8.67181013012378102479704402604335196876 2e-35L;

long double
atan2l(long double y, long double x)
{
        long double t, z;
        int k, m, signy, signx;

        if (x != x || y != y)
                return (x + y);         /* return NaN if x or y is NAN */

#endif /* ! codereview */
        signy = signbitl(y);
        signx = signbitl(x);

#endif /* ! codereview */
        if (x == one)
                return (atanl(y));

#endif /* ! codereview */
        /* Ensure sign indicators are boolean */
        m = (signy != 0) + (signx != 0) + (signx != 0);

        /* when y = 0 */
        if (y == zero) {
        if (y == zero)
                switch (m) {
                case 0:
                        return (y);             /* atan(+0,+anything) */
                case 1:
                        return (y);             /* atan(-0,+anything) */
                case 2:
                        return (PI + tiny);     /* atan(+0,-anything) */
                case 3:
                        return (-PI - tiny);    /* atan(-0,-anything) */
                }
        }
#endif /* ! codereview */

        /* when x = 0 */
        if (x == zero)
                return (signy != 0 ? -PIo2 - tiny : PIo2 + tiny);

        /* when x is INF */
        if (!finitel(x)) {
                if (!finitel(y)) {
                        switch (m) {
                        case 0:
                                return (PIo4 + tiny);   /* atan(+INF,+INF) */
                        case 1:
                                return (-PIo4 - tiny);  /* atan(-INF,+INF) */
                        case 2:
```

```
                                return (PI3o4 + tiny);  /* atan(+INF,-INF) */
                        case 3:
                                return (-PI3o4 - tiny); /* atan(-INF,-INF) */
                        }
                } else {
                        switch (m) {
                        case 0:
                                return (zero);          /* atan(+...,+INF) */
                        case 1:
                                return (-zero);         /* atan(-...,+INF) */
                        case 2:
                                return (PI + tiny);     /* atan(+...,-INF) */
                        case 3:
                                return (-PI - tiny);    /* atan(-...,-INF) */
                        }
                }
        }

#endif /* ! codereview */
        /* when y is INF */
        if (!finitel(y))
                return (signy != 0 ? -PIo2 - tiny : PIo2 + tiny);

        /* compute y/x */
        x = fabsl(x);
        y = fabsl(y);
        t = PI_lo;
        k = (ilogbl(y) - ilogbl(x));

        if (k > 120)
                z = PIo2 + half * t;
        else if (m > 1 && k < -120)
                z = zero;
        else
                z = atanl(y / x);

        switch (m) {
        case 0:
                return (z);             /* atan(+,+) */
        case 1:
                return (-z);            /* atan(-,+) */
        case 2:
                return (PI - (z - t));  /* atan(+,-) */
        case 3:
                return ((z - t) - PI);  /* atan(-,-) */
        }

#endif /* ! codereview */
        /* NOTREACHED */
        return (0.0L);
}
```

```
**********************************************************
    1229 Thu Jun  6 20:38:36 2019
new/usr/src/lib/libm/common/Q/atan2pil.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#include "libm.h"

/*
 * atan2pil(y,x) = atan2l(y, x) / pi
 */

static const long double invpi = 3.183098861837906715377675267450287240689e-1L;

long double
atan2pil(long double y, long double x)
{
atan2pil(long double y, long double x) {
        return (atan2l(y, x) * invpi);
}
______unchanged_portion_omitted_
```

```
**********************************************************
    1835 Thu Jun  6 20:38:37 2019
new/usr/src/lib/libm/common/Q/atanhl.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __atanhl = atanhl

#include "libm.h"

/*
 *                    1              2x                          x
 *      atanhl(x) = --- * LOG(1 + -------) = 0.5 * log1pl(2 * --------)
 *                    2            1 - x                       1 - x
 * Note: to guarantee atanhl(-x) = -atanhl(x), we use
 *                  sign(x)              |x|
 *      atanhl(x) = ------- * log1pl(2*-------).
 *                     2               1 - |x|
 *
 * Special cases:
 *      atanhl(x) is NaN if |x| > 1 with signal;
 *      atanhl(NaN) is that NaN with no signal;
 *      atanhl(+-1) is +-INF with signal.
 *
 */

static const long double zero = 0.0L, half = 0.5L, one = 1.0L;

long double
atanhl(long double x)
{
atanhl(long double x) {
        long double t;

        t = fabsl(x);
```



```
#endif /* ! codereview */
        if (t == one)
                return (x / zero);

#endif /* ! codereview */
        t = t / (one - t);
        return (copysignl(half, x) * log1pl(t + t));
}
```

```
**********************************************************
    6543 Thu Jun  6 20:38:37 2019
new/usr/src/lib/libm/common/Q/atanl.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __atanl = atanl

/*
 * atanl(x)
 * Table look-up algorithm
 * By K.C. Ng, March 9, 1989
 *
 * Algorithm.
 *
 * The algorithm is based on atan(x)=atan(y)+atan((x-y)/(1+x*y)).
 * We use poly1(x) to approximate atan(x) for x in [0,1/8] with
 * error (relative)
 *      |(atan(x)-poly1(x))/x|<= 2^-115.94      long double
 *      |(atan(x)-poly1(x))/x|<= 2^-58.85       double
 *      |(atan(x)-poly1(x))/x|<= 2^-25.53       float
 * and use poly2(x) to approximate atan(x) for x in [0,1/65] with
 * error (absolute)
 *      |atan(x)-poly2(x)|<= 2^-122.15          long double
 *      |atan(x)-poly2(x)|<= 2^-64.79           double
 *      |atan(x)-poly2(x)|<= 2^-35.36           float
 * Here poly1 and poly2 are odd polynomial with the following form:
 *              x + x^3*(a1+x^2*(a2+...))
 *
 * (0). Purge off Inf and NaN and 0
 * (1). Reduce x to positive by atan(x) = -atan(-x).
 * (2). For x <= 1/8, use
 *      (2.1) if x < 2^(-prec/2-2), atan(x) =  x  with inexact
 *      (2.2) Otherwise
 *              atan(x) = poly1(x)
 * (3). For x >= 8 then
```



```
 *      (3.1) if x >= 2^(prec+2),   atan(x) = atan(inf) - pio2lo
 *      (3.2) if x >= 2^(prec/3+2), atan(x) = atan(inf) - 1/x
 *      (3.3) if x >  65,           atan(x) = atan(inf) - poly2(1/x)
 *      (3.4) Otherwise,            atan(x) = atan(inf) - poly1(1/x)
 *
 * (4). Now x is in (0.125, 8)
 *      Find y that match x to 4.5 bit after binary (easy).
 *      If iy is the high word of y, then
 *              single : j = (iy - 0x3e000000) >> 19
 *              double : j = (iy - 0x3fc00000) >> 16
 *              quad   : j = (iy - 0x3ffc0000) >> 12
 *
 *      Let s = (x-y)/(1+x*y). Then
 *      atan(x) = atan(y) + poly1(s)
 *              = _TBL_atanl_hi[j] + (_TBL_atanl_lo[j] + poly2(s) )
 *
 *      Note. |s| <= 1.5384615385e-02 = 1/65. Maxium occurs at x = 1.03125
 *
 */

#include "libm.h"

extern const long double _TBL_atanl_hi[], _TBL_atanl_lo[];
static const long double one = 1.0L,
static const long double
        one     =   1.0L,
        p1 = -3.333333333333333333333333333331344526118e-0001L,
        p2 = 1.999999999999999999999999999989931277668570e-0001L,
        p3 = -1.428571428571428571428553606221309530901e-0001L,
        p4 = 1.111111111111111110952198427371397474 18e-0001L,
        p5 = -9.090909090909090825503603835248061123323e-0002L,
        p6 = 7.692307692307664052130743214708925258904e-0002L,
        p7 = -6.666666666660213835187713228363717388266e-0002L,
        p8 = 5.882352940152439399097283359608661949504e-0002L,
        p9 = -5.263157780447533993046614040509529668487e-0002L,
        p10 = 4.761895816878184933175855990886788439447e-0002L,
        p11 = -4.347345005832274022681019724553538135922e-0002L,
        p12 = 3.983031914579635037502589204647752042736e-0002L,
        p13 = -3.348206704469830575196657749413894897554e-0002L,
        q1 = -3.333333333333333333333333333319527365 0186e-0001L,
        q2 = 1.999999999999999999999988146114392615808e-0001L,
        q3 = -1.428571428571428571057630319435467111253e-0001L,
        q4 = 1.111111111111105373263048208994541544098e-0001L,
        q5 = -9.090909090421834209167373258681021816441e-0002L,
        q6 = 7.692305377813692706850171767150701644539e-0002L,
        q7 = -6.660896644393861499147317343057179013 30e-0002L,
        pio2hi = 1.570796326794896619231321691639751398740e+0000L,
        pio2lo = 4.335905065061890512398522013021675984381e-0035L;

#define i0      0
#define i1      3

long double
atanl(long double x)
{
atanl(long double x) {
        long double y, z, r, p, s;
        int *px = (int *)&x, *py = (int *)&y;
        int *px = (int *) &x, *py = (int *) &y;
        int ix, iy, sign, j;

        ix = px[i0];
        sign = ix & 0x80000000;
        ix ^= sign;

        /* for |x| < 1/8 */
```

```
124         if (ix < 0x3ffc0000) {
125                 if (ix < 0x3feb0000) {          /* when |x| < 2**(-prec/6-2) */
126                         if (ix < 0x3fc50000) {  /* if |x| < 2**(-prec/2-2) */
127                                 s = one;
128                                 *(3 - i0 + (int *)&s) = -1;     /* s = 1-ulp */
129                                 *(1 + (int *)&s) = -1;
130                                 *(2 + (int *)&s) = -1;
131                                 *(i0 + (int *)&s) -= 1;

133                                 if ((int)(s * x) < 1)
 68                                 *(3 - i0 + (int *) &s) = -1;    /* s = 1-ulp */
 69                                 *(1 + (int *) &s) = -1;
 70                                 *(2 + (int *) &s) = -1;
 71                                 *(i0 + (int *) &s) -= 1;
 72                                 if ((int) (s * x) < 1)
134                                         return (x);     /* raise inexact */
135                         }

137 #endif /* ! codereview */
138                         z = x * x;

140                         if (ix < 0x3fe20000)    /* if |x| < 2**(-prec/4-1) */
 75                         if (ix < 0x3fe20000) {  /* if |x| < 2**(-prec/4-1) */
141                                 return (x + (x * z) * p1);
142                         else                    /* if |x| < 2**(-prec/6-2) */
 77                         } else {        /* if |x| < 2**(-prec/6-2) */
143                                 return (x + (x * z) * (p1 + z * p2));
144                 }

 80                 }
146                 z = x * x;
147                 return (x + (x * z) * (p1 + z * (p2 + z * (p3 + z * (p4 + z *
148                     (p5 + z * (p6 + z * (p7 + z * (p8 + z * (p9 + z *
149                     (p10 + z * (p11 + z * (p12 + z * p13)))))))))))));
 82                 return (x + (x * z) * (p1 + z * (p2 + z * (p3 + z * (p4 +
 83                         z * (p5 + z * (p6 + z * (p7 + z * (p8 + z * (p9 +
 84                         z * (p10 + z * (p11 + z * (p12 + z * p13)))))))))))));
150         }

152         /* for |x| >= 8.0 */
153         if (ix >= 0x40020000) {
154                 px[i0] = ix;

156 #endif /* ! codereview */
157                 if (ix < 0x40050400) {  /* x <  65 */
158                         r = one / x;
159                         z = r * r;

161 #endif /* ! codereview */
162                         /*
163                          * poly1
164                          */
165                         y = r * (one + z * (p1 + z * (p2 + z * (p3 + z * (p4 +
166                             z * (p5 + z * (p6 + z * (p7 + z * (p8 + z * (p9 +
167                             z * (p10 + z * (p11 + z * (p12 + z *
168                             p13)))))))))))));
 90                         y = r * (one + z * (p1 + z * (p2 + z * (p3 +
 91                                 z * (p4 + z * (p5 + z * (p6 + z * (p7 +
 92                                 z * (p8 + z * (p9 + z * (p10 + z * (p11 +
 93                                 z * (p12 + z * p13)))))))))))));
169                         y -= pio2lo;
170                 } else if (ix < 0x40260000) {   /* x <  2**(prec/3+2) */
171                         r = one / x;
172                         z = r * r;

174 #endif /* ! codereview */
```



```
175                         /*
176                          * poly2
177                          */
178                         y = r * (one + z * (q1 + z * (q2 + z * (q3 + z * (q4 +
179                             z * (q5 + z * (q6 + z * q7)))))));
180                         y -= pio2lo;
181                 } else if (ix < 0x40720000) {   /* x <  2**(prec+2) */
182                         y = one / x - pio2lo;
183                 } else if (ix < 0x7fff0000) {   /* x <  inf */
184                         y = -pio2lo;
185                 } else {                        /* x is inf or NaN */
186                         if (((ix - 0x7fff0000) | px[1] | px[2] | px[i1]) != 0)
187                                 return (x - x);

189 #endif /* ! codereview */
190                         y = -pio2lo;
191                 }

193                 if (sign == 0)
194                         return (pio2hi - y);
195                 else
196                         return (y - pio2hi);
197         }

199         /* now x is between 1/8 and 8 */
200         px[i0] = ix;
201         iy = (ix + 0x00000800) & 0x7ffff000;
202         py[i0] = iy;
203         py[1] = py[2] = py[i1] = 0;
204         j = (iy - 0x3ffc0000) >> 12;

206         if (sign == 0)
207                 s = (x - y) / (one + x * y);
208         else
209                 s = (y - x) / (one + x * y);

211 #endif /* ! codereview */
212         z = s * s;

214 #endif /* ! codereview */
215         if (ix == iy)
216                 p = s * (one + z * (q1 + z * (q2 + z * (q3 + z * q4))));
217         else
218                 p = s * (one + z * (q1 + z * (q2 + z * (q3 + z * (q4 + z * (q5 +
219                     z * (q6 + z * q7)))))));

 98                 p = s * (one + z * (q1 + z * (q2 + z * (q3 + z * (q4 +
 99                         z * (q5 + z * (q6 + z * q7)))))));
221         if (sign == 0) {
222                 r = p + _TBL_atanl_lo[j];
223                 return (r + _TBL_atanl_hi[j]);
224         } else {
225                 r = p - _TBL_atanl_lo[j];
226                 return (r - _TBL_atanl_hi[j]);
227         }
228 }
```

_____*unchanged_portion_omitted_*

```
**********************************************************
    1681 Thu Jun  6 20:38:38 2019
new/usr/src/lib/libm/common/Q/cbrtl.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __cbrtl = cbrtl

#include "libm.h"
#include "longdouble.h"

#define n0      0

long double
cbrtl(long double x)
{
cbrtl(long double x) {
        long double s, t, r, w, y;
        double dx, dy;
        int *py = (int *)&dy;
        int *py = (int *) &dy;
        int n, m, m3, sx;

        if (!finitel(x))
                return (x + x);

#endif /* ! codereview */
        if (iszerol(x))
                return (x);

#endif /* ! codereview */
        sx = signbitl(x);
        x = fabsl(x);
        n = ilogbl(x);
        m = n / 3;
        m3 = m + m + m;
```



```
        y = scalbnl(x, -m3);
        dx = (double)y;
        dx = (double) y;
        dy = cbrt(dx);
        py[1 - n0] += 2;

#endif /* ! codereview */
        if (py[1 - n0] == 0)
                py[n0] += 1;

        /* one step newton iteration to 113 bits with error < 0.667ulps */
        t = (long double)dy;
        t = (long double) dy;
        t = scalbnl(t, m);
        s = t * t;
        r = x / s;
        w = t + t;
        r = (r - t) / (w + r);
        t += t * r;

        return (sx == 0 ? t : -t);
}
______unchanged_portion_omitted_
```

```
**********************************************************
    1236 Thu Jun  6 20:38:38 2019
new/usr/src/lib/libm/common/Q/copysignl.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __copysignl = copysignl

#include "libm.h"

long double
copysignl(long double x, long double y)
{
        int *px = (int *)&x;
        int *py = (int *)&y;
copysignl(long double x, long double y) {
        int *px = (int *) &x;
        int *py = (int *) &y;

        px[HIXWORD] = (px[HIXWORD] & ~XSGNMSK) | (py[HIXWORD] & XSGNMSK);
        return (x);
}
______unchanged_portion_omitted_
```

```
**********************************************************
    2936 Thu Jun  6 20:38:39 2019
new/usr/src/lib/libm/common/Q/coshl.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __coshl = coshl

#include "libm.h"
#include "longdouble.h"


/*
 * coshl(X)
 * RETURN THE HYPERBOLIC COSINE OF X
 *
 * Method :
 *      1. Replace x by |x| (coshl(x) = coshl(-x)).
 *      2.
 *                                                      [ expl(x) - 1 ]^2
 *          0        <= x <= 0.3465 : coshl(x) := 1 + -------------------
 *                                                            2*expl(x)
 *
 *                                              expl(x) + 1/expl(x)
 *          0.3465   <= x <= thresh : coshl(x) := -------------------
 *                                                         2
 *          thresh   <= x <= lnovft : coshl(x) := expl(x)/2
 *          lnovft   <= x <  INF    : coshl(x) := scalbnl(expl(x-1024*ln2),1023)
 *
 * here
 *      thr1            a number that is near one half of ln2.
 *      thr2            a number such that
 *                              expl(thresh)+expl(-thresh)=expl(thresh)
 *      lnovft:         logrithm of the overflow threshold
 *                      = MEP1*ln2 chopped to machine precision.
 *      ME              maximum exponent
```



```
 *      MEP1            maximum exponent plus 1
 *
 * Special cases:
 *      coshl(x) is |x| if x is +INF, -INF, or NaN.
 *      only coshl(0)=1 is exact for finite x.
 */

#define ME              16383
#define MEP1            16384
#define LNOVFT          1.13565234062941439494919310779707634284 5e+4L
/* last 32 bits of LN2HI is zero */
                /* last 32 bits of LN2HI is zero */
#define LN2HI           6.931471805599453094172319547495844850203e-0001L
#define LN2LO           1.667085920830552208890449330400379754169e-0025L
#define THR1            0.3465L
#define THR2            45.L

static const long double half = 0.5L,
static const long double
        half    = 0.5L,
        tinyl = 7.5e-37L,
        one = 1.0L,
        ln2hi = LN2HI,
        ln2lo = LN2LO,
        lnovftL = LNOVFT,
        thr1 = THR1,
        thr2 = THR2;

long double
coshl(long double x)
{
coshl(long double x) {
        long double t, w;

        w = fabsl(x);

#endif /* ! codereview */
        if (!finitel(w))
                return (w + w);         /* x is INF or NaN */

#endif /* ! codereview */
        if (w < thr1) {
                t = w < tinyl ? w : expm1l(w);
                w = one + t;

#endif /* ! codereview */
                if (w != one)
                        w = one + (t * t) / (w + w);

#endif /* ! codereview */
                return (w);
        } else if (w < thr2) {
                t = expl(w);
                return (half * (t + one / t));
        } else if (w <= lnovftL) {
        } else if (w <= lnovftL)
                return (half * expl(w));
        } else {
        else {
                return (scalbnl(expl((w - MEP1 * ln2hi) - MEP1 * ln2lo), ME));
        }
}
______unchanged_portion_omitted_
```

```
**********************************************************
    2675 Thu Jun  6 20:38:39 2019
new/usr/src/lib/libm/common/Q/cosl.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

/*
 * cosl(x)
 * Table look-up algorithm by K.C. Ng, November, 1989.
 *
 * kernel function:
 *      __k_sinl        ... sin function on [-pi/4,pi/4]
 *      __k_cosl        ... cos function on [-pi/4,pi/4]
 *      __rem_pio2l     ... argument reduction routine
 *
 * Method.
 *      Let S and C denote the sin and cos respectively on [-PI/4, +PI/4].
 *      1. Assume the argument x is reduced to y1+y2 = x-k*pi/2 in
 *         [-pi/2 , +pi/2], and let n = k mod 4.
 *      2. Let S=S(y1+y2), C=C(y1+y2). Depending on n, we have
 *
 *          n        sin(x)      cos(x)        tan(x)
 *     ----------------------------------------------------------
 *          0          S           C             S/C
 *          1          C          -S            -C/S
 *          2         -S          -C             S/C
 *          3         -C           S            -C/S
 *     ----------------------------------------------------------
 *
 * Special cases:
 *      Let trig be any of sin, cos, or tan.
 *      trig(+-INF)  is NaN, with signals;
 *      trig(NaN)    is that NaN;
 *
 * Accuracy:
 *      computer TRIG(x) returns trig(x) nearly rounded.
```



```
 */

#pragma weak __cosl = cosl

#include "libm.h"
#include "longdouble.h"

long double
cosl(long double x)
{
cosl(long double x) {
        long double y[2], z = 0.0L;
        int n, ix;

        ix = *(int *)&x;                /* High word of x */
        ix = *(int *) &x;                       /* High word of x */

        ix &= 0x7fffffff;

        if (ix <= 0x3ffe9220) {         /* |x| ~< pi/4 */
        if (ix <= 0x3ffe9220)                   /* |x| ~< pi/4 */
                return (__k_cosl(x, z));
        } else if (ix >= 0x7fff0000) {  /* trig(Inf or NaN) is NaN */
        else if (ix >= 0x7fff0000)              /* trig(Inf or NaN) is NaN */
                return (x - x);
        } else {                        /* argument reduction needed */
        else {                                  /* argument reduction needed */
                n = __rem_pio2l(x, y);

#endif /* ! codereview */
                switch (n & 3) {
                case 0:
                        return (__k_cosl(y[0], y[1]));
                case 1:
                        return (-__k_sinl(y[0], y[1]));
                case 2:
                        return (-__k_cosl(y[0], y[1]));
                case 3:
                        return (__k_sinl(y[0], y[1]));
                }
        }

#endif /* ! codereview */
        /* NOTREACHED */
        return (0.0L);
    return 0.0L;
}
```

_____*unchanged_portion_omitted_*

```
**********************************************************
   13505 Thu Jun  6 20:38:40 2019
new/usr/src/lib/libm/common/Q/erfl.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

/*
 * long double function erf,erfc (long double x)
 * K.C. Ng, September, 1989.
 *                            x
 *                   2        |\
 *     erf(x)  =  ---------   | exp(-t*t)dt
 *                sqrt(pi) \|
 *                            0
 *
 *     erfc(x) =  1-erf(x)
 *
 * method:
 *      Since erf(-x) = -erf(x), we assume x>=0.
 *      For x near 0, we have the expansion
 *
 *          erf(x) = (2/sqrt(pi))*(x - x^3/3 + x^5/10 - x^7/42 + ....).
 *
 *      Since 2/sqrt(pi) = 1.128379167095512573896158903121545171688,
 *      we use x + x*P(x^2) to approximate erf(x). This formula will
 *      guarantee the error less than one ulp where x is not too far
 *      away from 0. We note that erf(x)=x at x = 0.6174..... After
 *      some experiment, we choose the following approximation on
 *      interval [0,0.84375].
 *
 *      For x in [0,0.84375]
 *                2                2        4                40
 *         P =  P(x ) = (p0 + p1 * x + p2 * x + ... + p20 * x  )
 *
 *         erf(x)  = x + x*P
 *         erfc(x) = 1 - erf(x)           if x<=0.25
```



```
 *                 = 0.5 + ((0.5-x)-x*P)  if x in [0.25,0.84375]
 *      precision: |P(x^2)-(erf(x)-x)/x| <= 2**-122.50
 *
 *      For x in [0.84375,1.25], let s = x - 1, and
 *      c = 0.84506291151 rounded to single (24 bits)
 *         erf(x)  = c  + P1(s)/Q1(s)
 *         erfc(x) = (1-c)  - P1(s)/Q1(s)
 *      precision: |P1/Q1 - (erf(x)-c)| <= 2**-118.41
 *
 *
 *      For x in [1.25,1.75], let s = x - 1.5, and
 *      c = 0.95478588343 rounded to single (24 bits)
 *         erf(x)  = c  + P2(s)/Q2(s)
 *         erfc(x) = (1-c)  - P2(s)/Q2(s)
 *      precision: |P1/Q1 - (erf(x)-c)| <= 2**-123.83
 *
 *
 *      For x in [1.75,16/3]
 *         erfc(x) = exp(-x*x)*(1/x)*R1(1/x)/S1(1/x)
 *         erf(x)  = 1 - erfc(x)
 *      precision: absolute error of R1/S1 is bounded by 2**-124.03
 *
 *      For x in [16/3,107]
 *         erfc(x) = exp(-x*x)*(1/x)*R2(1/x)/S2(1/x)
 *         erf(x)  = 1 - erfc(x) (if x>=9 simple return erf(x)=1 with inexact)
 *      precision: absolute error of R2/S2 is bounded by 2**-120.07
 *
 *      Else if inf > x >= 107
 *         erf(x)  = 1 with inexact
 *         erfc(x) = 0 with underflow
 *
 *      Special case:
 *         erf(inf)  = 1
 *         erfc(inf) = 0
 */

#pragma weak __erfl = erfl
#pragma weak __erfcl = erfcl

#include "libm.h"
#include "longdouble.h"

static const long double tiny = 1e-40L,
static const long double
        tiny        = 1e-40L,
        nearunfl = 1e-4000L,
        half = 0.5L,
        one = 1.0L,
        onehalf = 1.5L,
        L16_3 = 16.0L / 3.0L;

        L16_3       = 16.0L/3.0L;
/*
 * Coefficients for even polynomial P for erf(x)=x+x*P(x^2) on [0,0.84375]
 */
static const long double P[] = {        /* 21 coeffs */
        1.283791670955125738961589031215451715556e-0001L,
        -3.761263890318375246320529677071815594603e-0001L,
        1.128379167095512573896158903121205899135e-0001L,
        -2.686617064513125175943235483344625046092e-0002L,
        5.223977625442187842111846652980454568389e-0003L,
        -8.548327023450852832546626271083862724358e-0004L,
        1.205533298178966425102164715902231976672e-0004L,
        -1.492565035840625097674944905027897838996e-0005L,
        1.646211436588924733604648849172936692024e-0006L,
        -1.636584469123491976815834704799733514987e-0007L,
```

```
        1.48071928158789744530252900714477073930 5e-0008L,
        -1.22905553017078284304646798646472204717 5e-0009L,
        9.42275906432030735755395494576065434163 3e-0011L,
        -6.71136684665343903616210510499143338092 6e-0012L,
        4.46322409034189316510027538069384311624 0e-0013L,
        -2.78351345258265824542263566255977916231 2e-0014L,
        1.63422741258696019525134687886375466154 6e-0015L,
        -9.06078267288957772276571145562311780279 5e-0017L,
        4.74134180126624687341215921389361360235 4e-0018L,
        -2.27241759649782618837484663653431738120 3e-0019L,
        8.06908873371606846249683565892856692093 3e-0021L,
};
```

*unchanged_portion_omitted_*

```
#endif /* ! codereview */
static const long double Q1[] = { /* 12 bottom coeffs with leading 1.0 hidden */
        9.08150629606488219528017837310762319665 5e-0001L,
        6.82104953196820409760439218365068764252 0e-0001L,
        4.06786917823353950231505597074327182283 8e-0001L,
        1.70233223354631676581814472306388109557 7e-0001L,
        7.49809837769055393426642308870861421935 6e-0002L,
        2.05015439691817869705692723436637276031 0e-0002L,
        7.01298853403199989905478233385190593937 9e-0003L,
        1.14990478701440035464984345123457073107 6e-0003L,
        3.18562025501129947619603949120515971862 0e-0004L,
        1.27340507215300877542637619337410584051 7e-0005L,
        4.75386699995943297195678122814840297145 4e-0006L,
        -1.00228760211166002605398172854954020068 3e-0006L,
};

#endif /* ! codereview */
/*
 * Rational erf(x) = ((float)0.95478588343) + P2(x-1.5)/Q2(x-1.5)
 * on [1.25,1.75]
 */
static const long double C2 = (long double)((float)0.95478588343);
static const long double P2[] = {       /*  12 top coeffs */
        1.13192630486444673013512616459478586351 2e-0002L,
        1.27361799696775415154433005518621032283 2e-0001L,
        -8.16998073466751251989781690719028114342 3e-0002L,
        9.51226748609032119783363427178794427174 6e-0002L,
        -2.39425156980487216000527499973591436817 0e-0002L,
        1.10876866022752866752525233318452022290 5e-0002L,
        3.52743549293390241466204331437327749422 1e-0004L,
        4.94611627334195346358431900666947462597 1e-0004L,
        -4.28985194251314471460028576902242096241 8e-0005L,
        8.30471984134195270587478163600208511997 8e-0005L,
        -1.04046022617730933878190225228284990318 9e-0005L,
        2.12291333158492147038132758367204443408 7e-0006L,
};

#endif /* ! codereview */
static const long double Q2[] = { /* 13 bottom coeffs with leading 1.0 hidden */
        7.44881573730699274916872769104200383215 0e-0001L,
        7.16181385023600829448474431243012218804 3e-0001L,
        3.60313475658422576614492272740564123612 1e-0001L,
        1.95581160913376647808055079519453585265 3e-0001L,
        7.25305996371622597247969381378781071123 3e-0002L,
        2.75239125375742142421277022154123832497 8e-0002L,
        7.67765485208524025743905067344654682800 5e-0003L,
        2.14110224455550968734649706032663006106 9e-0003L,
        4.34212301383095709394956333913067436427 1e-0004L,
        8.66458789557004334853099199727221215031 6e-0005L,
        1.10920158251175208706016742939703370198 8e-0005L,
        1.35783437578183106271334700003098436431 1e-0006L,
        4.95774628059438499727309038506068001645 1e-0008L,
```



```
};

#endif /* ! codereview */
/*
 * erfc(x) = exp(-x*x)/x * R1(1/x)/S1(1/x) on [1.75, 16/3]
 */
static const long double R1[] = {       /*  14 top coeffs */
        4.63019512265431501637070576762155060294 8e+0006L,
        1.25794952174649483070065420448867571362 8e+0007L,
        1.70415382272026027281474349737618162570 7e+0007L,
        1.50260056870606187238157753953731573994 3e+0007L,
        9.54371079343199528482702444538733392286 1e+0006L,
        4.58934480858409101165223816493594952242 7e+0006L,
        1.71466066294174579119090707192067184428 9e+0006L,
        5.03480214776879889430767225619246628386 7e+0005L,
        1.16228640044355467055315211044712685072 5e+0005L,
        2.08664383454890168136275730805866039913 7e+0004L,
        2.83979316186814030590700439289034877733 8e+0003L,
        2.78668724165842360177825869449865568077 8e+0002L,
        1.77917783710269560242589745262398578646 4e+0001L,
        5.64189583547747076904361462381914443473 1e-0001L,
};

#endif /* ! codereview */
static const long double S1[] = { /* 15 bottom coeffs with leading 1.0 hidden */
        4.63019512265433152959560689628759684311 0e+0006L,
        1.78041109334551204324781084220509055058 e+0007L,
        3.25011309705180070370710862371577684828 3e+0007L,
        3.73785709917675505091219371212348911575 5e+0007L,
        3.02978749751657882145917405587078116859 3e+0007L,
        1.83385061996538476500576963210320577722 7e+0007L,
        8.56271999973691572221039122263918658649 8e+0006L,
        3.13968456207465897131554553976000813697 3e+0006L,
        9.10642131373138488002770362745436693094 5e+0005L,
        2.08510834238426650861326713600319492000 1e+0005L,
        3.72312627269312034073049141644953929060 0e+0004L,
        5.04916987856734404614569536078443692980 2e+0003L,
        4.94427453274801076767015073003539209389 9e+0002L,
        3.15351060881821392998294024916226897141 2e+0001L,
        1.0e00L,
};

/*
 * erfc(x) = exp(-x*x)/x * R2(1/x)/S2(1/x) on [16/3, 107]
 */
static const long double R2[] = { /*  15 top coeffs in reverse order!! */
static const long double R2[] = {       /*  15 top coeffs in reverse order!!*/
        2.44728801225430296679632658753713693166 9e+0005L,
        8.76859256718986189665336991271653873901 6e+0005L,
        1.55229315258178006576149790800577952495 3e+0006L,
        1.79207592483594293586423165750425992672 9e+0006L,
        1.50400146315589734494750022205269483587 5e+0006L,
        9.69948555632689141180123018601601301993 5e+0005L,
        4.96144993366180796986343501336479603770 0e+0005L,
        2.04872654469347402806117676471622827379 1e+0005L,
        6.89153296433094972247906109055189688663 5e+0004L,
        1.88801470901030750777196404790582323798 5e+0004L,
        4.18969206498895774505473480964249564450 2e+0003L,
        7.36234648742704806821296888964274173462 1e+0002L,
        9.98035971421141142300764105658081311620 7e+0001L,
        9.42691089513537918110719196219348517415 9e+0000L,
        5.64189583547756286948079451562360128042 9e-0001L,
};

#endif /* ! codereview */
static const long double S2[] = {       /* 16 coefficients */
```

```
        2.447282203601902971246004716790604686880e+0005L,
        1.153009852759385309367759460934808489833e+0006L,
        2.608580649612639131548966265078663384849e+0006L,
        3.766673917346623308850202792390569025740e+0006L,
        3.890566255138383910789924920541335370691e+0006L,
        3.052882073900746207613166259994150527732e+0006L,
        1.885574519970380988460241047248519418407e+0006L,
        9.369722034759943185851450846811445012922e+0005L,
        3.792278350536686111444869752624492443659e+0005L,
        1.257750606950115799965366001773094058720e+0005L,
        3.410830600242369370645608634643620355058e+0004L,
        7.513984469742343134851326863175067271240e+0003L,
        1.313296320593190002554779998138695507840e+0003L,
        1.773972700887629157006326333696896516769e+0002L,
        1.670876451822586800422009013880457094162e+0001L,
        1.000L,
};

long double
erfl(long double x)
long double erfl(x)
long double x;
{
        long double s, y, t;
        long double s,y,t;

        if (!finitel(x)) {
                if (x != x)
                        return (x + x);         /* NaN */

                return (copysignl(one, x));     /* return +-1.0 is x=Inf */
            if (x != x) return x+x;     /* NaN */
            return copysignl(one,x);    /* return +-1.0 is x=Inf */
        }

        y = fabsl(x);

#endif /* ! codereview */
        if (y <= 0.84375L) {
                if (y <= tiny)
                        return (x + P[0] * x);

                s = y * y;
                t = __poly_libmq(s, 21, P);
                return (x + x * t);
            if (y<=tiny) return x+P[0]*x;
            s = y*y;
            t = __poly_libmq(s,21,P);
            return  x+x*t;
        }

        if (y <= 1.25L) {
                s = y - one;
                t = C1 + __poly_libmq(s, 12, P1) / (one + s * __poly_libmq(s,
                    12, Q1));
                return ((signbitl(x)) ? -t : t);
        } else if (y <= 1.75L) {
                s = y - onehalf;
                t = C2 + __poly_libmq(s, 12, P2) / (one + s * __poly_libmq(s,
                    13, Q2));
                return ((signbitl(x)) ? -t : t);
        if (y<=1.25L) {
            s = y-one;
            t = C1+__poly_libmq(s,12,P1)/(one+s*__poly_libmq(s,12,Q1));
            return (signbitl(x))? -t: t;
        } else if (y<=1.75L) {
```



```
            s = y-onehalf;
            t = C2+__poly_libmq(s,12,P2)/(one+s*__poly_libmq(s,13,Q2));
            return (signbitl(x))? -t: t;
        }

        if (y <= 9.0L)
                t = erfcl(y);
        else
                t = tiny;

        return ((signbitl(x)) ? t - one : one - t);
        if (y<=9.0L) t = erfcl(y); else t = tiny;
        return (signbitl(x))? t-one: one-t;
}

long double
erfcl(long double x)
long double erfcl(x)
long double x;
{
        long double s, y, t;
        long double s,y,t;

        if (!finitel(x)) {
                if (x != x)
                        return (x + x);         /* NaN */

            if (x != x) return x+x;     /* NaN */
                /* return 2.0 if x= -inf; 0.0 if x= +inf */
                if (x < 0.0L)
                        return (2.0L);
                else
                        return (0.0L);
            if (x < 0.0L) return 2.0L; else return 0.0L;
        }

        if (x <= 0.84375L) {
                if (x <= 0.25)
                        return (one - erfl(x));

                s = x * x;
                t = half - x;
                t = t - x * __poly_libmq(s, 21, P);
                return (half + t);
            if (x<=0.25) return one-erfl(x);
            s = x*x;
            t = half-x;
            t = t - x*__poly_libmq(s,21,P);
            return  half+t;
        }

        if (x <= 1.25L) {
                s = x - one;
                t = one - C1;
                return (t - __poly_libmq(s, 12, P1) / (one + s * __poly_libmq(s,
                    12, Q1)));
        } else if (x <= 1.75L) {
                s = x - onehalf;
                t = one - C2;
                return (t - __poly_libmq(s, 12, P2) / (one + s * __poly_libmq(s,
                    13, Q2)));
        if (x<=1.25L) {
            s = x-one;
            t = one-C1;
            return t - __poly_libmq(s,12,P1)/(one+s*__poly_libmq(s,12,Q1));
        } else if (x<=1.75L) {
```

```
            s = x-onehalf;
            t = one-C2;
            return t - __poly_libmq(s,12,P2)/(one+s*__poly_libmq(s,13,Q2));
        }

        if (x >= 107.0L) {
                return (nearunfl * nearunfl);   /* underflow */
        } else if (x >= L16_3) {
                y = __poly_libmq(x, 15, R2);
                t = y / __poly_libmq(x, 16, S2);
        if (x>=107.0L) return nearunfl*nearunfl;                /* underflow */
        else if (x >= L16_3) {
            y = __poly_libmq(x,15,R2);
            t = y/__poly_libmq(x,16,S2);
        } else {
                y = __poly_libmq(x, 14, R1);
                t = y / __poly_libmq(x, 15, S1);
            y = __poly_libmq(x,14,R1);
            t = y/__poly_libmq(x,15,S1);
        }

#endif /* ! codereview */
        /*
         * Note that exp(-x*x+d) = exp(-x*x)*exp(d), so to compute
         * exp(-x*x) with a small relative error, we need to compute
         * -x*x with a small absolute error.  To this end, we set y
         * equal to the leading part of x but with enough trailing
         * zeros that y*y can be computed exactly and we rewrite x*x
         * as y*y + (x-y)*(x+y), distributing the latter expression
         * across the exponential.
         *
         * We could construct y in a portable way by setting
         *
         *   int i = (int)(x * ptwo);
         *   y = (long double)i * 1/ptwo;
         *
         * where ptwo is some power of two large enough to make x-y
         * small but not so large that the conversion to int overflows.
         * When long double arithmetic is slow, however, the following
         * non-portable code is preferable.
         */
        y = x;
        *(2 + (int *)&y) = *(3 + (int *)&y) = 0;
        t *= expl(-y * y) * expl(-(x - y) * (x + y));
        return (t);
        *(2+(int*)&y) = *(3+(int*)&y) = 0;
        t *= expl(-y*y)*expl(-(x-y)*(x+y));
        return  t;
}
```

_____*unchanged_portion_omitted_*

```
**********************************************************
    2868 Thu Jun  6 20:38:40 2019
new/usr/src/lib/libm/common/Q/exp10l.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
   1 /*
   2  * CDDL HEADER START
   3  *
   4  * The contents of this file are subject to the terms of the
   5  * Common Development and Distribution License (the "License").
   6  * You may not use this file except in compliance with the License.
   7  *
   8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
   9  * or http://www.opensolaris.org/os/licensing.
  10  * See the License for the specific language governing permissions
  11  * and limitations under the License.
  12  *
  13  * When distributing Covered Code, include this CDDL HEADER in each
  14  * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  15  * If applicable, add the following below this CDDL HEADER, with the
  16  * fields enclosed by brackets "[]" replaced with your own identifying
  17  * information: Portions Copyright [yyyy] [name of copyright owner]
  18  *
  19  * CDDL HEADER END
  20  */

  22 /*
  23  * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
  24  */

  26 #endif /* ! codereview */
  27 /*
  28  * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
  29  * Use is subject to license terms.
  30  */

  32 #include "libm.h"
  33 #include "longdouble.h"

  35 /*
  36  * exp10l(x)
  37  *      n = nint(x*(log10/log2)) ;
  38  *      exp10(x) = 10**x = exp(x*ln(10)) = exp(n*ln2+(x*ln10-n*ln2))
  39  *               = 2**n*exp(ln10*(x-n*log2/log10)))
  40  *      If x is an integer <= M then use repeat multiplication. For
  41  *      10**M is the largest representable integer, where
  42  *              M = 10          single precision (24 bits)
  43  *              M = 22          double precision (53 bits)
  44  *              M = 48          quadruple precision (113 bits)
  45  */

  47 #define TINY    1.0e-20L        /* single: 1e-5, double: 1e-10, quad: 1e-20 */
  48 #define LG10OVT 4933.L          /* single:  39, double:  309, quad:  4933 */
  49 #define LG10UFT -4966.L         /* single: -45, double: -323, quad: -4966 */
  50 #define M               48
  51 /* logt2hi : last 32 bits is zero for quad prec */
  25                         /* logt2hi : last 32 bits is zero for quad prec */
  52 #define LOGT2HI         0.301029995663981195213738894724209860346 88L
  53 #define LOGT2LO         2.831664213089468167896664371953e-31L

  55 static const long double zero = 0.0L,
  29 static const long double
  30         zero    = 0.0L,
  56         tiny = TINY * TINY,
  57         one = 1.0L,
  58         lg10 = 3.321928094887362347870319429489390175865e+0000L,
```



```
  59         ln10 = 2.302585092994045684017991454684364207601e+0000L,
  60         logt2hi = LOGT2HI,
  61         logt2lo = LOGT2LO,
  62         lg10ovt = LG10OVT,
  63         lg10uft = LG10UFT;

  65 long double
  66 exp10l(long double x)
  67 {
  41 exp10l(long double x) {
  68         long double t, tenp;
  69         int k;

  71         if (!finitel(x)) {
  72                 if (isnanl(x) || x > zero)
  73                         return (x + x);
  74                 else
  75                         return (zero);
  76         }

  78 #endif /* ! codereview */
  79         if (fabsl(x) < tiny)
  80                 return (one + x);

  82         if (x <= lg10ovt) {
  51         if (x <= lg10ovt)
  83                 if (x >= lg10uft) {
  84                         k = (int)x;
  53                         k = (int) x;
  85                         tenp = 10.0L;

  87 #endif /* ! codereview */
  88                         /* x is a small +integer */
  89                         if (0 <= k && k <= M && (long double)k == x) {
  55                         if (0 <= k && k <= M && (long double) k == x) {
  90                                 t = one;

  92 #endif /* ! codereview */
  93                                 if (k & 1)
  94                                         t *= tenp;

  96 #endif /* ! codereview */
  97                                 k >>= 1;

  99 #endif /* ! codereview */
 100                                 while (k) {
 101                                         tenp *= tenp;

 103 #endif /* ! codereview */
 104                                         if (k & 1)
 105                                                 t *= tenp;

 107 #endif /* ! codereview */
 108                                         k >>= 1;
 109                                 }

 111 #endif /* ! codereview */
 112                                 return (t);
 113                         }

 115 #endif /* ! codereview */
 116                         t = anintl(x * lg10);
 117                         return (scalbnl(expl(ln10 * ((x - t * logt2hi) - t *
 118                             logt2lo)), (int)t));
 119                 } else {
  57                         return (scalbnl(expl(ln10 * ((x - t * logt2hi) -
```

```
                                t * logt2lo)), (int) t));
                } else
                        return (scalbnl(one, -50000));  /* underflow */
                }
        } else {
        else
                return (scalbnl(one, 50000));           /* overflow  */
        }
#endif /* ! codereview */
}
```

```
**********************************************************
    2104 Thu Jun  6 20:38:40 2019
new/usr/src/lib/libm/common/Q/exp2l.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __exp2l = exp2l

#include "libm.h"
#include "longdouble.h"

/*
 *      exp2l(x) = 2**x = 2**((x-anint(x))+anint(x))
 *               = 2**anint(x)*2**(x-anint(x))
 *               = 2**anint(x)*exp((x-anint(x))*ln2)
 */

#define TINY    1.0e-20L        /* single: 1e-5, double: 1e-10, quad: 1e-20 */
#define OVFLEXP 16400           /* single: 130,  double  1030,  quad: 16400 */
#define UNFLEXP -16520          /* single:-155,  double -1080,  quad:-16520 */

static const long double zero = 0.0L,
static const long double
        zero = 0.0L,
        tiny = TINY * TINY,
        half = 0.5L,
        ln2 = 6.931471805599453094172321214581765680755e-0001L,
        one = 1.0L;

static const int ovflexp = OVFLEXP, unflexp = UNFLEXP;
static const int
        ovflexp = OVFLEXP,
        unflexp = UNFLEXP;

long double
exp2l(long double x)
```



```
{
exp2l(long double x) {
        long double t;

        if (!finitel(x)) {
                if (isnanl(x) || x > zero)
                        return (x + x);
                else
                        return (zero);
        }

#endif /* ! codereview */
        t = fabsl(x);

#endif /* ! codereview */
        if (t < half) {
                if (t < tiny)
                        return (one + x);
                else
                        return (expl(ln2 * x));
        }

#endif /* ! codereview */
        t = anintl(x);

#endif /* ! codereview */
        if (t < ovflexp) {
                if (t >= unflexp)
                        return (scalbnl(expl(ln2 * (x - t)), (int)t));
                        return (scalbnl(expl(ln2 * (x - t)), (int) t));
                else
                        return (scalbnl(one, unflexp));         /* underflow */
        } else {
        } else
                return (scalbnl(one, ovflexp));                 /* overflow  */
        }
#endif /* ! codereview */
}
```

```
**********************************************************
    3921 Thu Jun  6 20:38:41 2019
new/usr/src/lib/libm/common/Q/expl.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

/*
 * expl(x)
 * Table driven method
 * Written by K.C. Ng, November 1988.
 * Algorithm :
 *      1. Argument Reduction: given the input x, find r and integer k
 *         and j such that
 *                    x = (32k+j)*ln2 + r,  |r| <= (1/64)*ln2 .
 *
 *      2. expl(x) = 2^k * (2^(j/32) + 2^(j/32)*expm1(r))
 *         Note:
 *         a. expm1(r) = (2r)/(2-R), R = r - r^2*(t1 + t2*r^2)
 *         b. 2^(j/32) is represented as
 *                       _TBL_expl_hi[j]+_TBL_expl_lo[j]
 *         where
 *              _TBL_expl_hi[j] = 2^(j/32) rounded
 *              _TBL_expl_lo[j] = 2^(j/32) - _TBL_expl_hi[j].
 *
 * Special cases:
 *      expl(INF) is INF, expl(NaN) is NaN;
 *      expl(-INF)=  0;
 *      for finite argument, only expl(0)=1 is exact.
 *
 * Accuracy:
 *      according to an error analysis, the error is always less than
 *      an ulp (unit in the last place).
 *
 * Misc. info.
 *      For 113 bit long double
 *              if x >  1.135652340629414394949193107797076342845e+4
```



```
 *      then expl(x) overflow;
 *              if x < -1.143346274333629787883724384345262150341e+4
 *      then expl(x) underflow
 *
 * Constants:
 * Only decimal values are given. We assume that the compiler will convert
 * from decimal to binary accurately enough to produce the correct
 * hexadecimal values.
 */

#pragma weak __expl = expl

#include "libm.h"

extern const long double _TBL_expl_hi[], _TBL_expl_lo[];
static const long double one = 1.0L,
        two = 2.0L,
        ln2_64 = 1.083042469624914545964425189778400898568e-2L,
        ovflthreshold = 1.135652340629414394949193107797076342845e+4L,
        unflthreshold = -1.143346274333629787883724384345262150341e+4L,
        invln2_32 = 4.616624130844682903551758979206054839765e+1L,
        ln2_32hi = 2.166084939249829091928849858592451515688e-2L,
        ln2_32lo = 5.209643502595475652782654157501186731779e-27L;

static const long double
one             =  1.0L,
two             =  2.0L,
ln2_64          =  1.083042469624914545964425189778400898568e-2L,
ovflthreshold   =  1.135652340629414394949193107797076342845e+4L,
unflthreshold   = -1.143346274333629787883724384345262150341e+4L,
invln2_32       =  4.616624130844682903551758979206054839765e+1L,
ln2_32hi        =  2.166084939249829091928849858592451515688e-2L,
ln2_32lo        =  5.209643502595475652782654157501186731779e-27L;

/* rational approximation coeffs for [-(ln2)/64,(ln2)/64] */
static const long double t1 = 1.666666666666666666666666666660876387437e-1L,
        t2 = -2.777777777777777777777778128093173478756e-3L,
        t3 = 6.613756613756613482074280932874221202424e-5L,
        t4 = -1.653439153392139954169609822742235851120e-6L,
        t5 = 4.175314851769539751387852116610973796053e-8L;
static const long double
t1 =   1.666666666666666666666666666660876387437e-1L,
t2 =  -2.777777777777777777777778128093173478756e-3L,
t3 =   6.613756613756613482074280932874221202424e-5L,
t4 =  -1.653439153392139954169609822742235851120e-6L,
t5 =   4.175314851769539751387852116610973796053e-8L;

long double
expl(long double x)
{
        int *px = (int *)&x, ix, j, k, m;
expl(long double x) {
        int *px = (int *) &x, ix, j, k, m;
        long double t, r;

        ix = px[0];                     /* high word of x */

#endif /* ! codereview */
        if (ix >= 0x7fff0000)
                return (x + x);         /* NaN of +inf */

        if (((unsigned)ix) >= 0xffff0000)
        if (((unsigned) ix) >= 0xffff0000)
                return (-one / x);      /* NaN or -inf */

#endif /* ! codereview */
```

```
        if ((ix & 0x7fffffff) < 0x3fc30000) {
                if ((int)x < 1)
                if ((int) x < 1)
                        return (one + x);       /* |x|<2^-60 */
        }

#endif /* ! codereview */
        if (ix > 0) {
                if (x > ovflthreshold)
                        return (scalbnl(x, 20000));

                k = (int)(invln2_32 * (x + ln2_64));
                k = (int) (invln2_32 * (x + ln2_64));
        } else {
                if (x < unflthreshold)
                        return (scalbnl(-x, -40000));

                k = (int)(invln2_32 * (x - ln2_64));
                k = (int) (invln2_32 * (x - ln2_64));
        }

        j = k & 0x1f;
        m = k >> 5;
        t = (long double)k;
        j  = k&0x1f;
        m  = k>>5;
        t  = (long double) k;
        x = (x - t * ln2_32hi) - t * ln2_32lo;
        t = x * x;
        r = (x - t * (t1 + t * (t2 + t * (t3 + t * (t4 + t * t5))))) - two;
        x = _TBL_expl_hi[j] - ((_TBL_expl_hi[j] * (x + x)) / r -
            _TBL_expl_lo[j]);
        return (scalbnl(x, m));
}
```

_____***unchanged_portion_omitted_***

```
**********************************************************
    6019 Thu Jun  6 20:38:41 2019
new/usr/src/lib/libm/common/Q/expm1l.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __expm1l = expm1l

#if !defined(__sparc)
#error Unsupported architecture
#endif

/*
 * expm1l(x)
 *
 * Table driven method
 * Written by K.C. Ng, June 1995.
 * Algorithm :
 *      1. expm1(x) = x if x<2**-114
 *      2. if |x| <= 0.0625 = 1/16, use approximation
 *              expm1(x) = x + x*P/(2-P)
 * where
 *      P = x - z*(P1+z*(P2+z*(P3+z*(P4+z*(P5+z*P6+z*P7))))), z = x*x;
 * (this formula is derived from
 *      2-P+x = R = x*(exp(x)+1)/(exp(x)-1) ~ 2 + x*x/6 - x^4/360 + ...)
 *
 * P1 =   1.66666666666666666666666666666638500528074603030e-0001
 * P2 =  -2.77777777777777777777777759668391122822266551158e-0003
 * P3 =   6.61375661375661375657437408890138814721051293054e-0005
 * P4 =  -1.65343915343915303310185228411892601606669528828e-0006
 * P5 =   4.17535139755122945763580609663414647067443411178e-0008
 * P6 =  -1.05683795988668526689182102605260986731620026832e-0009
 * P7 =   2.67544168821852702827123344217198187229611470514e-0011
 *
 * Accuracy: |R-x*(exp(x)+1)/(exp(x)-1)|<=2**-119.13
 *
```



```
 *      3. For 1/16 < |x| < 1.125, choose x(+-i) ~ +-(i+4.5)/64, i=0,..,67
 *         since
 *              exp(x) = exp(xi+(x-xi))= exp(xi)*exp((x-xi))
 *         we have
 *              expm1(x) = expm1(xi)+(exp(xi))*(expm1(x-xi))
 *         where
 *              |s=x-xi| <= 1/128
 *         and
 *      expm1(s)=2s/(2-R), R= s-s^2*(T1+s^2*(T2+s^2*(T3+s^2*(T4+s^2*T5))))
 *
 * T1 =   1.66666666666666666666666666660876387437e-1L,
 * T2 =  -2.77777777777777777777777078120931734787 56e-3L,
 * T3 =   6.61375661375661348207428093287422120242 4e-5L,
 * T4 =  -1.65343915339213995416960982274223585112 0e-6L,
 * T5 =   4.17531485176953975138785211661097379605 3e-8L;
 *
 *      4. For |x| >= 1.125, return exp(x)-1.
 *          (see algorithm for exp)
 *
 * Special cases:
 *      expm1l(INF) is INF, expm1l(NaN) is NaN;
 *      expm1l(-INF)= -1;
 *      for finite argument, only expm1l(0)=0 is exact.
 *
 * Accuracy:
 *      according to an error analysis, the error is always less than
 *      2 ulp (unit in the last place).
 *
 * Misc. info.
 *      For 113 bit long double
 *              if x >  1.13565234062941439494919310779707634284 5e+4
 *      then expm1l(x) overflow;
 *
 * Constants:
 * Only decimal values are given. We assume that the compiler will convert
 * from decimal to binary accurately enough to produce the correct
 * hexadecimal values.
 */

#include "libm.h"

extern const long double _TBL_expl_hi[], _TBL_expl_lo[];
extern const long double _TBL_expm1lx[], _TBL_expm1l[];
static const long double zero = +0.0L,

static const long double
        zero            = +0.0L,
        one = +1.0L,
        two = +2.0L,
        ln2_64 = +1.08304246962491454596442518977840089856 8e-2L,
        ovflthreshold = +1.13565234062941439494919310779707634284 5e+4L,
        invln2_32 = +4.61662413084468290355175897920605483976 5e+1L,
        ln2_32hi = +2.16608493924982909192884985859245151568 8e-2L,
        ln2_32lo = +5.20964350259547565278265415750118673177 9e-27L,
        huge = +1.0e4000L,
        tiny = +1.0e-4000L,
        P1 = +1.66666666666666666666666666666638500528074603030e-0001L,
        P2 = -2.77777777777777777777777596683911228222665511 58e-0003L,
        P3 = +6.61375661375661375657437408890138814721051293054e-0005L,
        P4 = -1.65343915343915303310185228411892601606669528828e-0006L,
        P5 = +4.17535139755122945763580609663414647067443411178e-0008L,
        P6 = -1.05683795988668526689182102605260986731620026832e-0009L,
        P7 = +2.67544168821852702827123344217198187229611470514e-0011L,
/* rational approximation coeffs for [-(ln2)/64,(ln2)/64] */
        T1 = +1.66666666666666666666666666660876387437e-1L,
        T2 = -2.77777777777777777777777078120931734787 56e-3L,
```

```
        T3 = +6.61375661375661348207428093287422120242 4e-5L,
        T4 = -1.65343915339213995416960982274223585112 0e-6L,
        T5 = +4.17531485176953975138785211661097379605 3e-8L;

long double
expm1l(long double x)
{
expm1l(long double x) {
        int hx, ix, j, k, m;
        long double t, r, s, w;

        hx = ((int *)&x)[HIXWORD];
        hx = ((int *) &x)[HIXWORD];
        ix = hx & ~0x80000000;

#endif /* ! codereview */
        if (ix >= 0x7fff0000) {
                if (x != x)
                        return (x + x);                 /* NaN */

#endif /* ! codereview */
                if (x < zero)
                        return (-one);                  /* -inf */

#endif /* ! codereview */
                return (x);                             /* +inf */
        }

#endif /* ! codereview */
        if (ix < 0x3fff4000) {                          /* |x| < 1.25 */
                if (ix < 0x3ffb0000) {                  /* |x| < 0.0625 */
                        if (ix < 0x3f8d0000) {
                                if ((int)x == 0)
                                if ((int) x == 0)
                                        return (x);     /* |x|<2^-114 */
                        }

#endif /* ! codereview */
                        t = x * x;
                        r = (x - t * (P1 + t * (P2 + t * (P3 + t * (P4 + t *
                            (P5 + t * (P6 + t * P7)))))));
                        return (x + (x * r) / (two - r));
                }

#endif /* ! codereview */
                /* compute i = [64*x] */
                m = 0x4009 - (ix >> 16);
                j = ((ix & 0x0000ffff) | 0x10000) >> m; /* j=4,...,67 */

#endif /* ! codereview */
                if (hx < 0)
                        j += 82;                        /* negative */

#endif /* ! codereview */
                s = x - _TBL_expm1lx[j];
                t = s * s;
                r = s - t * (T1 + t * (T2 + t * (T3 + t * (T4 + t * T5))));
                r = (s + s) / (two - r);
                w = _TBL_expm1l[j];
                return (w + (w + one) * r);
        }

#endif /* ! codereview */
        if (hx > 0) {
                if (x > ovflthreshold)
                        return (huge * huge);
```



```
                k = (int)(invln2_32 * (x + ln2_64));
                k = (int) (invln2_32 * (x + ln2_64));
        } else {
                if (x < -80.0)
                        return (tiny - x / x);

                k = (int)(invln2_32 * (x - ln2_64));
                k = (int) (invln2_32 * (x - ln2_64));
        }

#endif /* ! codereview */
        j = k & 0x1f;
        m = k >> 5;
        t = (long double)k;
        t = (long double) k;
        x = (x - t * ln2_32hi) - t * ln2_32lo;
        t = x * x;
        r = (x - t * (T1 + t * (T2 + t * (T3 + t * (T4 + t * T5))))) - two;
        x = _TBL_expl_hi[j] - ((_TBL_expl_hi[j] * (x + x)) / r -
            _TBL_expl_lo[j]);
        return (scalbnl(x, m) - one);
}
```

*_____unchanged_portion_omitted_*

```
**********************************************************
    1142 Thu Jun  6 20:38:42 2019
new/usr/src/lib/libm/common/Q/fabsl.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __fabsl = fabsl

#include "libm.h"

long double
fabsl(long double x)
{
        int *px = (int *)&x;
fabsl(long double x) {
        int *px = (int *) &x;

        px[0] &= 0x7fffffff;
        return (x);
}
______unchanged_portion_omitted_
```

```
**********************************************************
    1413 Thu Jun  6 20:38:42 2019
new/usr/src/lib/libm/common/Q/finitel.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#include "libm.h"

#if defined(__sparc)
int
finitel(long double x)
{
        int *px = (int *)&x;

finitel(long double x) {
        int *px = (int *) &x;
        return ((px[0] & ~0x80000000) < 0x7fff0000);
}
#elif defined(__x86)
int
finitel(long double x)
{
        int *px = (int *)&x, t = px[2] & 0x7fff;

finitel(long double x) {
        int *px = (int *) &x, t = px[2] & 0x7fff;
#if defined(HANDLE_UNSUPPORTED)
        return (t != 0x7fff && ((px[1] & 0x80000000) != 0 || t == 0));
#else
        return (t != 0x7fff);
#endif
}
______unchanged_portion_omitted_
```

```
**********************************************************
    1848 Thu Jun  6 20:38:43 2019
new/usr/src/lib/libm/common/Q/floorl.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

/*
 * ceill(x)     return the biggest integral value below x
 * floorl(x)    return the least integral value above x
 *
 * NOTE: aintl(x), anintl(x), ceill(x), floorl(x), and rintl(x) return result
 * with the same sign as x's,  including 0.0.
 */

#pragma weak __ceill = ceill
#pragma weak __floorl = floorl

#include "libm.h"
#include "longdouble.h"

static const long double qone = 1.0L;

long double
ceill(long double x)
{
ceill(long double x) {
        long double t;

        if (!finitel(x))
                return (x + x);

#endif /* ! codereview */
        t = rintl(x);

#endif /* ! codereview */
        if (t >= x)                     /* already ceil(x) */
```



```
                return (t);
        else                            /* t < x case: return t+1  */
                return (copysignl(t + qone, x));
}

long double
floorl(long double x)
{
floorl(long double x) {
        long double t;

        if (!finitel(x))
                return (x + x);

#endif /* ! codereview */
        t = rintl(x);

#endif /* ! codereview */
        if (t <= x)
                return (t);             /* already floor(x) */
        else                            /* x < t case: return t-1  */
                return (copysignl(t - qone, x));
}
```

```
**********************************************************
    5796 Thu Jun  6 20:38:43 2019
new/usr/src/lib/libm/common/Q/fmodl.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __fmodl = fmodl

#include "libm.h"

static const int is = -0x7fffffff - 1, im = 0x0000ffff, iu = 0x00010000;
static const long double zero = 0.0L, one = 1.0L;
static const int
        is = -0x7fffffff - 1,
        im = 0x0000ffff,
        iu = 0x00010000;

static const long double
        zero = 0.0L,
        one = 1.0L;

#ifdef __LITTLE_ENDIAN
#define __H0(x)         *(3 + (int *)&x)
#define __H1(x)         *(2 + (int *)&x)
#define __H2(x)         *(1 + (int *)&x)
#define __H3(x)         *(0 + (int *)&x)
#define __H0(x) *(3 + (int *) &x)
#define __H1(x) *(2 + (int *) &x)
#define __H2(x) *(1 + (int *) &x)
#define __H3(x) *(0 + (int *) &x)
#else
#define __H0(x)         *(0 + (int *)&x)
#define __H1(x)         *(1 + (int *)&x)
#define __H2(x)         *(2 + (int *)&x)
#define __H3(x)         *(3 + (int *)&x)
#define __H0(x) *(0 + (int *) &x)
```



```
#define __H1(x) *(1 + (int *) &x)
#define __H2(x) *(2 + (int *) &x)
#define __H3(x) *(3 + (int *) &x)
#endif

long double
fmodl(long double x, long double y)
{
fmodl(long double x, long double y) {
        long double a, b;
        int n, ix, iy, k, sx;
        int hx;
        int x0, y0, z0, carry;
        unsigned x1, x2, x3, y1, y2, y3, z1, z2, z3;

        hx = __H0(x);
        x1 = __H1(x);
        x2 = __H2(x);
        x3 = __H3(x);
        y0 = __H0(y);
        y1 = __H1(y);
        y2 = __H2(y);
        y3 = __H3(y);

        sx = hx & 0x80000000;
        x0 = hx ^ sx;
        y0 &= 0x7fffffff;

        /* purge off exception values */
        if (x0 >= 0x7fff0000 ||         /* !finitel(x) */
            (y0 > 0x7fff0000) || (y0 == 0x7fff0000 && ((y1 | y2 | y3) != 0)) ||
            (y0 | y1 | y2 | y3) == 0)   /* isnanl(y) || y = 0 */
                return ((x * y) / (x * y));

#endif /* ! codereview */
        a = fabsl(x);
        b = fabsl(y);

#endif /* ! codereview */
        if (a <= b) {
                if (a < b)
                        return (x);
                else
                        return (zero * x);
        }

#endif /* ! codereview */
        /* determine ix = ilogbl(x) */
        if (x0 < iu) {                  /* subnormal x */
                ix = -16382;

#endif /* ! codereview */
                while (x0 == 0) {
                        ix -= 16;
                        x0 = x1 >> 16;
                        x1 = (x1 << 16) | (x2 >> 16);
                        x2 = (x2 << 16) | (x3 >> 16);
                        x3 = (x3 << 16);
                }

#endif /* ! codereview */
                while (x0 < iu) {
                        ix -= 1;
                        x0 = (x0 << 1) | (x1 >> 31);
                        x1 = (x1 << 1) | (x2 >> 31);
                        x2 = (x2 << 1) | (x3 >> 31);
```

```
                        x3 <<= 1;
                }
        } else {
                ix = (x0 >> 16) - 16383;
                x0 = iu | (x0 & im);
        }

        /* determine iy = ilogbl(y) */
        if (y0 < iu) {                  /* subnormal y */
                iy = -16382;

#endif /* ! codereview */
                while (y0 == 0) {
                        iy -= 16;
                        y0 = y1 >> 16;
                        y1 = (y1 << 16) | (y2 >> 16);
                        y2 = (y2 << 16) | (y3 >> 16);
                        y3 = (y3 << 16);
                }

#endif /* ! codereview */
                while (y0 < iu) {
                        iy -= 1;
                        y0 = (y0 << 1) | (y1 >> 31);
                        y1 = (y1 << 1) | (y2 >> 31);
                        y2 = (y2 << 1) | (y3 >> 31);
                        y3 <<= 1;
                }
        } else {
                iy = (y0 >> 16) - 16383;
                y0 = iu | (y0 & im);
        }

        /* fix point fmod */
        n = ix - iy;

#endif /* ! codereview */
        while (n--) {
                while (x0 == 0 && n >= 16) {
                        n -= 16;
                        x0 = x1 >> 16;
                        x1 = (x1 << 16) | (x2 >> 16);
                        x2 = (x2 << 16) | (x3 >> 16);
                        x3 = (x3 << 16);
                }

#endif /* ! codereview */
                while (x0 < iu && n >= 1) {
                        n -= 1;
                        x0 = (x0 << 1) | (x1 >> 31);
                        x1 = (x1 << 1) | (x2 >> 31);
                        x2 = (x2 << 1) | (x3 >> 31);
                        x3 = (x3 << 1);
                }

#endif /* ! codereview */
                carry = 0;
                z3 = x3 - y3;
                carry = (z3 > x3);

#endif /* ! codereview */
                if (carry == 0) {
                        z2 = x2 - y2;
                        carry = (z2 > x2);
                } else {
                        z2 = x2 - y2 - 1;
```



```
                        carry = (z2 >= x2);
                }

#endif /* ! codereview */
                if (carry == 0) {
                        z1 = x1 - y1;
                        carry = (z1 > x1);
                } else {
                        z1 = x1 - y1 - 1;
                        carry = (z1 >= x1);
                }

#endif /* ! codereview */
                z0 = x0 - y0 - carry;

#endif /* ! codereview */
                if (z0 < 0) {           /* double x */
                        x0 = x0 + x0 + ((x1 & is) != 0);
                        x1 = x1 + x1 + ((x2 & is) != 0);
                        x2 = x2 + x2 + ((x3 & is) != 0);
                        x3 = x3 + x3;
                } else {
                        if (z0 == 0) {
                                if ((z1 | z2 | z3) == 0) {      /* 0: done */
                                        __H0(a) = hx & is;
                                        __H1(a) = __H2(a) = __H3(a) = 0;
                                        return (a);
                                }
                        }

#endif /* ! codereview */
                        /* x = z << 1 */
                        z0 = z0 + z0 + ((z1 & is) != 0);
                        z1 = z1 + z1 + ((z2 & is) != 0);
                        z2 = z2 + z2 + ((z3 & is) != 0);
                        z3 = z3 + z3;
                        x0 = z0;
                        x1 = z1;
                        x2 = z2;
                        x3 = z3;
                }
        }

        carry = 0;
        z3 = x3 - y3;
        carry = (z3 > x3);

#endif /* ! codereview */
        if (carry == 0) {
                z2 = x2 - y2;
                carry = (z2 > x2);
        } else {
                z2 = x2 - y2 - 1;
                carry = (z2 >= x2);
        }

#endif /* ! codereview */
        if (carry == 0) {
                z1 = x1 - y1;
                carry = (z1 > x1);
        } else {
                z1 = x1 - y1 - 1;
                carry = (z1 >= x1);
        }

#endif /* ! codereview */
```

```
        z0 = x0 - y0 - carry;

#endif /* ! codereview */
        if (z0 >= 0) {
                x0 = z0;
                x1 = z1;
                x2 = z2;
                x3 = z3;
        }

#endif /* ! codereview */
        /* convert back to floating value and restore the sign */
        if ((x0 | x1 | x2 | x3) == 0) {
                __H0(a) = hx & is;
                __H1(a) = __H2(a) = __H3(a) = 0;
                return (a);
        }

#endif /* ! codereview */
        while (x0 < iu) {
                if (x0 == 0) {
                        iy -= 16;
                        x0 = x1 >> 16;
                        x1 = (x1 << 16) | (x2 >> 16);
                        x2 = (x2 << 16) | (x3 >> 16);
                        x3 = (x3 << 16);
                } else {
                        x0 = x0 + x0 + ((x1 & is) != 0);
                        x1 = x1 + x1 + ((x2 & is) != 0);
                        x2 = x2 + x2 + ((x3 & is) != 0);
                        x3 = x3 + x3;
                        iy -= 1;
                }
        }

        /* normalize output */
        if (iy >= -16382) {
                __H0(a) = sx | (x0 - iu) | ((iy + 16383) << 16);
                __H1(a) = x1;
                __H2(a) = x2;
                __H3(a) = x3;
        } else {                        /* subnormal output */
                n = -16382 - iy;
                k = n & 31;

#endif /* ! codereview */
                if (k != 0) {
                        if (k <= 16) {
                                x3 = (x2 << (32 - k)) | (x3 >> k);
                                x2 = (x1 << (32 - k)) | (x2 >> k);
                                x1 = (x0 << (32 - k)) | (x1 >> k);
                                x0 >>= k;
                        } else {
                                x3 = (x2 << (32 - k)) | (x3 >> k);
                                x2 = (x1 << (32 - k)) | (x2 >> k);
                                x1 = (x0 << (32 - k)) | (x1 >> k);
                                x0 = 0;
                        }
                }

#endif /* ! codereview */
                while (n >= 32) {
                        n -= 32;
                        x3 = x2;
                        x2 = x1;
                        x1 = x0;
```



```
                        x0 = 0;
                }

#endif /* ! codereview */
                __H0(a) = x0 | sx;
                __H1(a) = x1;
                __H2(a) = x2;
                __H3(a) = x3;
                a *= one;
        }

#endif /* ! codereview */
        return (a);
}
```

```
**********************************************************
    1321 Thu Jun  6 20:38:44 2019
new/usr/src/lib/libm/common/Q/gammal.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __gammal = gammal

/*
 * long double gammal(long double x);
 */

#include "libm.h"
#include "longdouble.h"

extern int signgam;
extern int signgaml;

long double
gammal(long double x)
{
gammal(long double x) {
        long double y = __k_lgammal(x, &signgaml);

        signgam = signgaml;     /* SUSv3 requires the setting of signgam */
        return (y);
}
_____unchanged_portion_omitted_
```

```
**********************************************************
    1234 Thu Jun  6 20:38:44 2019
new/usr/src/lib/libm/common/Q/gammal_r.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

/*
 * long double gammal_r(long double x, int *signgamlp);
 */

#pragma weak __gammal_r = gammal_r

#include "libm.h"
#include "longdouble.h"

long double
gammal_r(long double x, int *signgamlp)
{
gammal_r(long double x, int *signgamlp) {
        return (__k_lgammal(x, signgamlp));
}
______unchanged_portion_omitted_
```

```
**********************************************************
    3914 Thu Jun  6 20:38:45 2019
new/usr/src/lib/libm/common/Q/hypotl.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __hypotl = hypotl

/*
 * long double hypotl(long double x, long double y);
 * Method :
 *      If z=x*x+y*y has error less than sqrt(2)/2 ulp than sqrt(z) has
 *      error less than 1 ulp.
 *      So, compute sqrt(x*x+y*y) with some care as follows:
 *      Assume x>y>0;
 *      1. save and set rounding to round-to-nearest
 *      2. if x > 2y  use
 *              x1*x1+(y*y+(x2*(x+x2))) for x*x+y*y
 *      where x1 = x with lower 64 bits cleared, x2 = x-x1; else
 *      3. if x <= 2y use
 *              t1*y1+((x-y)*(x-y)+(t1*y2+t2*y))
 *      where t1 = 2x with lower 64 bits cleared, t2 = 2x-t1, y1= y with
 *      lower 64 bits chopped, y2 = y-y1.
 *
 *      NOTE: DO NOT remove parenthsis!
 *
 * Special cases:
 *      hypot(x,y) is INF if x or y is +INF or -INF; else
 *      hypot(x,y) is NAN if x or y is NAN.
 *
 * Accuracy:
 *      hypot(x,y) returns sqrt(x^2+y^2) with error less than 1 ulps (units
 *      in the last place)
 */

#include "libm.h"
```



```
#include "longdouble.h"

extern enum fp_direction_type __swapRD(enum fp_direction_type);

static const long double zero = 0.0L, one = 1.0L;

long double
hypotl(long double x, long double y)
{
hypotl(long double x, long double y) {
        int n0, n1, n2, n3;
        long double t1, t2, y1, y2, w;
        int *px = (int *)&x, *py = (int *)&y;
        int *pt1 = (int *)&t1, *py1 = (int *)&y1;
        int *px = (int *) &x, *py = (int *) &y;
        int *pt1 = (int *) &t1, *py1 = (int *) &y1;
        enum fp_direction_type rd;
        int j, k, nx, ny, nz;

        if ((*(int *)&one) != 0) {      /* determine word ordering */
        if ((*(int *) &one) != 0) {     /* determine word ordering */
                n0 = 0;
                n1 = 1;
                n2 = 2;
                n3 = 3;
        } else {
                n0 = 3;
                n1 = 2;
                n2 = 1;
                n3 = 0;
        }

        px[n0] &= 0x7fffffff;           /* clear sign bit of x and y */
        py[n0] &= 0x7fffffff;
        k = 0x7fff0000;
        nx = px[n0] & k;                /* exponent of x and y */
        ny = py[n0] & k;

#endif /* ! codereview */
        if (ny > nx) {
                w = x;
                x = y;
                y = w;
                nz = ny;
                ny = nx;
                nx = nz;
        }                               /* force x > y */

#endif /* ! codereview */
        if ((nx - ny) >= 0x00730000)
                return (x + y); /* x/y >= 2**116 */

#endif /* ! codereview */
        if (nx < 0x5ff30000 && ny > 0x205b0000) {       /* medium x,y */
                /* save and set RD to Rounding to nearest */
                rd = __swapRD(fp_nearest);
                w = x - y;

#endif /* ! codereview */
                if (w > y) {
                        pt1[n0] = px[n0];
                        pt1[n1] = px[n1];
                        pt1[n2] = pt1[n3] = 0;
                        t2 = x - t1;
                        x = sqrtl(t1 * t1 - (y * (-y) - t2 * (x + t1)));
                } else {
```

```
                        x = x + x;
                        py1[n0] = py[n0];
                        py1[n1] = py[n1];
                        py1[n2] = py1[n3] = 0;
                        y2 = y - y1;
                        pt1[n0] = px[n0];
                        pt1[n1] = px[n1];
                        pt1[n2] = pt1[n3] = 0;
                        t2 = x - t1;
                        x = sqrtl(t1 * y1 - (w * (-w) - (t2 * y1 + y2 * x)));
                }

#endif /* ! codereview */
                if (rd != fp_nearest)
                        (void) __swapRD(rd);    /* restore rounding mode */

#endif /* ! codereview */
                return (x);
        } else {
                if (nx == k || ny == k) {       /* x or y is INF or NaN */
                        if (isinfl(x))
                                t2 = x;
                        else if (isinfl(y))
                                t2 = y;
                        else
                                t2 = x + y;     /* invalid if x or y is sNaN */

#endif /* ! codereview */
                        return (t2);
                }

#endif /* ! codereview */
                if (ny == 0) {
                        if (y == zero || x == zero)
                                return (x + y);

#endif /* ! codereview */
                        t1 = scalbnl(one, 16381);
                        x *= t1;
                        y *= t1;
                        return (scalbnl(one, -16381) * hypotl(x, y));
                }

#endif /* ! codereview */
                j = nx - 0x3fff0000;
                px[n0] -= j;
                py[n0] -= j;
                pt1[n0] = nx;
                pt1[n1] = pt1[n2] = pt1[n3] = 0;
                return (t1 * hypotl(x, y));
        }
}
```

```
**********************************************************
    2562 Thu Jun  6 20:38:45 2019
new/usr/src/lib/libm/common/Q/ieee_funcl.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#include "libm.h"

#if defined(__sparc)
int
isinfl(long double x)
{
        int *px = (int *)&x;

        return ((px[0] & ~0x80000000) == 0x7fff0000 && px[1] == 0 && px[2] ==
            0 && px[3] == 0);
isinfl(long double x) {
        int *px = (int *) &x;
        return ((px[0] & ~0x80000000) == 0x7fff0000 && px[1] == 0 &&
                px[2] == 0 && px[3] == 0);
}

int
isnormall(long double x)
{
        int *px = (int *)&x;

        return ((unsigned)((px[0] & 0x7fff0000) - 0x10000) < 0x7ffe0000);
isnormall(long double x) {
        int *px = (int *) &x;
        return ((unsigned) ((px[0] & 0x7fff0000) - 0x10000) < 0x7ffe0000);
}

int
issubnormall(long double x)
{
```



```
        int *px = (int *)&x;

issubnormall(long double x) {
        int *px = (int *) &x;
        px[0] &= ~0x80000000;
        return (px[0] < 0x00010000 && (px[0] | px[1] | px[2] | px[3]) != 0);
}

int
iszerol(long double x)
{
        int *px = (int *)&x;

iszerol(long double x) {
        int *px = (int *) &x;
        return (((px[0] & ~0x80000000) | px[1] | px[2] | px[3]) == 0);
}

int
signbitl(long double x)
{
        unsigned *px = (unsigned *)&x;

signbitl(long double x) {
        unsigned *px = (unsigned *) &x;
        return (px[0] >> 31);
}
#elif defined(__x86)
int
isinfl(long double x)
{
        int *px = (int *)&x;

isinfl(long double x) {
        int *px = (int *) &x;
#if defined(HANDLE_UNSUPPORTED)
        return ((px[2] & 0x7fff) == 0x7fff && ((px[1] ^ 0x80000000) | px[0]) ==
            0);
        return ((px[2] & 0x7fff) == 0x7fff &&
                ((px[1] ^ 0x80000000) | px[0]) == 0);
#else
        return ((px[2] & 0x7fff) == 0x7fff && ((px[1] & ~0x80000000) | px[0]) ==
            0);
        return ((px[2] & 0x7fff) == 0x7fff &&
                ((px[1] & ~0x80000000) | px[0]) == 0);
#endif
}

int
isnormall(long double x)
{
        int *px = (int *)&x;

isnormall(long double x) {
        int *px = (int *) &x;
#if defined(HANDLE_UNSUPPORTED)
        return ((unsigned)((px[2] & 0x7fff) - 1) < 0x7ffe && (px[1] &
            0x80000000) != 0);
        return ((unsigned) ((px[2] & 0x7fff) - 1) < 0x7ffe &&
                (px[1] & 0x80000000) != 0);
#else
        return ((unsigned)((px[2] & 0x7fff) - 1) < 0x7ffe);
        return ((unsigned) ((px[2] & 0x7fff) - 1) < 0x7ffe);
#endif
}
```

```
int
issubnormall(long double x)
{
        int *px = (int *)&x;

issubnormall(long double x) {
        int *px = (int *) &x;
        return ((px[2] & 0x7fff) == 0 && (px[0] | px[1]) != 0);
}

int
iszerol(long double x)
{
        int *px = (int *)&x;

iszerol(long double x) {
        int *px = (int *) &x;
        return (((px[2] & 0x7fff) | px[0] | px[1]) == 0);
}

int
signbitl(long double x)
{
        unsigned *px = (unsigned *)&x;

signbitl(long double x) {
        unsigned *px = (unsigned *) &x;
        return ((px[2] >> 15) & 1);
}
```

_____***unchanged_portion_omitted_***

```
**********************************************************
    2463 Thu Jun  6 20:38:46 2019
new/usr/src/lib/libm/common/Q/ilogbl.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __ilogbl = ilogbl

#include "libm.h"
#include "xpg6.h"                               /* __xpg6 */

#if defined(__sparc)
#define ISNORMALL(k, x)         (k != 0x7fff)   /* assuming k != 0 */
#define X86PDNRM(k, x)
#define XSCALE_OFFSET           0x406f          /* 0x3fff + 112 */

#endif /* ! codereview */
static const long double xscale = 5192296858534827628530496329220096.0L;

/* 2^112 */
                                                                        /* 2^112 */
#elif defined(__x86)
/*
 * if pseudo-denormal, replace by the equivalent normal
 */
#define X86PDNRM(k, x)  \
        if (k == 0 && (((int *)&x)[1] & 0x80000000) !=  0) \
                ((int *)&x)[2] |= k = 1
#define X86PDNRM(k, x)  if (k == 0 && (((int *) &x)[1] & 0x80000000) != 0) \
                                ((int *) &x)[2] |= k = 1
#if defined(HANDLE_UNSUPPORTED)         /* assuming k != 0 */
#define ISNORMALL(k, x) (k != 0x7fff && (((int *)&x)[1] & 0x80000000) != \
        0)
#define ISNORMALL(k, x) (k != 0x7fff && (((int *) &x)[1] & 0x80000000) != 0)
#else
#define ISNORMALL(k, x)         (k != 0x7fff)
```



```
#endif
#define XSCALE_OFFSET   0x403e                          /* 0x3fff + 63 */
static const long double xscale = 9223372036854775808.0L;       /* 2^63 */
#endif

static int
raise_invalid(int v)   /* SUSv3 requires ilogbl(0,+/-Inf,NaN) raise invalid */
{
raise_invalid(int v) {  /* SUSv3 requires ilogbl(0,+/-Inf,NaN) raise invalid */
#ifndef lint
        if ((__xpg6 & _C99SUSv3_ilogb_0InfNaN_raises_invalid) != 0) {
                static const double zero = 0.0;
                volatile double dummy;

                dummy = zero / zero;
        }
#endif
        return (v);
}

int
ilogbl(long double x)
{
ilogbl(long double x) {
        int k = XBIASED_EXP(x);

        X86PDNRM(k, x);

#endif /* ! codereview */
        if (k == 0) {
                if (ISZEROL(x)) {
                if (ISZEROL(x))
                        return (raise_invalid(0x80000001));
                } else {
                else {
                        x *= xscale;    /* scale up by 2^112 or 2^63 */
                        return (XBIASED_EXP(x) - XSCALE_OFFSET);
                }
        } else if (ISNORMALL(k, x)) {
        } else if (ISNORMALL(k, x))
                return (k - 0x3fff);
        } else {
        else
                return (raise_invalid(0x7fffffff));
        }
#endif /* ! codereview */
}
```

```
**********************************************************
    1588 Thu Jun  6 20:38:46 2019
new/usr/src/lib/libm/common/Q/isnanl.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __isnanl = isnanl

#include "libm.h"

#if defined(__sparc)
int
isnanl(long double x)
{
        int *px = (int *)&x;

        return ((px[0] & ~0x80000000) >= 0x7fff0000 && ((px[0] & ~0xffff0000) |
            px[1] | px[2] | px[3]) != 0);
isnanl(long double x) {
        int *px = (int *) &x;
        return ((px[0] & ~0x80000000) >= 0x7fff0000 &&
                ((px[0] & ~0xffff0000) | px[1] | px[2] | px[3]) != 0);
}
#elif defined(__x86)
int
isnanl(long double x)
{
        int *px = (int *)&x, t = px[2] & 0x7fff;

isnanl(long double x) {
        int *px = (int *) &x, t = px[2] & 0x7fff;
#if defined(HANDLE_UNSUPPORTED)
        return (t == 0x7fff && ((px[1] & ~0x80000000) | px[0]) != 0 || t != 0 &&
            (px[1] & 0x80000000) == 0);
        return (t == 0x7fff && ((px[1] & ~0x80000000) | px[0]) != 0 ||
                t != 0 && (px[1] & 0x80000000) == 0);
```



```
#else
        return (t == 0x7fff && ((px[1] & ~0x80000000) | px[0]) != 0);
#endif
}
```

_____*unchanged_portion_omitted_*

```
**********************************************************
   27749 Thu Jun  6 20:38:47 2019
new/usr/src/lib/libm/common/Q/j0l.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

/*
 * Floating point Bessel's function of the first and second kinds
 * of order zero: j0(x),y0(x);
 *
 * Special cases:
 *      y0(0)=y1(0)=yn(n,0) = -inf with division by zero signal;
 *      y0(-ve)=y1(-ve)=yn(n,-ve) are NaN with invalid signal.
 */

#pragma weak __j0l = j0l
#pragma weak __y0l = y0l

#include "libm.h"
#include "longdouble.h"

#define GENERIC long double

static const GENERIC zero = 0.0L,
        small = 1.0e-9L,
        tiny = 1.0e-38L,
        one = 1.0L,
        five = 5.0L,
        eight = 8.0L,
        invsqrtpi = 5.641895835477562869480794515607725858441e-0001L,
        tpi = 0.636619772367581343075535053490057448L;
static const GENERIC
zero    = 0.0L,
small   = 1.0e-9L,
tiny    = 1.0e-38L,
one     = 1.0L,
```



```
five    = 5.0L,
eight   = 8.0L,
invsqrtpi= 5.641895835477562869480794515607725858441e-0001L,
tpi     = 0.636619772367581343075535053490057448L;

static GENERIC pzero(GENERIC);
static GENERIC qzero(GENERIC);

static GENERIC r0[7] = {
        -2.499999999999999999999999999999998934492e-0001L,
        1.272657927360049786327618451133763714880e-0002L,
        -2.694499763712963276900636693400659600898e-0004L,
        2.724877475058977576903234070919616447883e-0006L,
        -1.432617103214330236967477495393076320281e-0008L,
        3.823248804080079168706683540513792224471e-0011L,
        -4.183174277567983647337568504286313665065e-0014L,
};

#endif /* ! codereview */
static GENERIC s0[7] = {
        1.0e0L,
        1.159368290559800854689526195462884666395e-0002L,
        6.629397597394973383009743876169946772559e-0005L,
        2.426779981394054406305431142501735094340e-0007L,
        6.097663491248511069094400469635449749883e-0010L,
        1.017019133340929220238747413216052224036e-0012L,
        9.012593179306197579518374581969371278481e-0016L,
};

GENERIC
j0l(GENERIC x)
{
        GENERIC z, s, c, ss, cc, r, u, v;
j0l(x) GENERIC x;{
        GENERIC z, s,c,ss,cc,r,u,v;
        int i;

        if (isnanl(x))
                return (x + x);

        if (isnanl(x)) return x+x;
        x = fabsl(x);

#endif /* ! codereview */
        if (x > 1.28L) {
                if (!finitel(x))
                        return (zero);

                if (!finitel(x)) return zero;
                s = sinl(x);
                c = cosl(x);

                /* BEGIN CSTYLED */
#endif /* ! codereview */
                /*
                 * j0(x) = sqrt(2/(pi*x))*(p0(x)*cos(x0)-q0(x)*sin(x0))
                 * where x0 = x-pi/4
                 *      Better formula:
                 *              cos(x0) = cos(x)cos(pi/4)+sin(x)sin(pi/4)
                 *                      = 1/sqrt(2) * (cos(x) + sin(x))
                 *              sin(x0) = sin(x)cos(pi/4)-cos(x)sin(pi/4)
                 *                      = 1/sqrt(2) * (sin(x) - cos(x))
                 * To avoid cancellation, use
                 *              sin(x) +- cos(x) = -cos(2x)/(sin(x) -+ cos(x))
                 * to compute the worse one.
                 */
```

```
115                 /* END CSTYLED */
116                 if (x > 1.0e2450L) {    /* x+x may overflow */
117                         ss = s - c;
118                         cc = s + c;
 56                 if (x>1.0e2450L) {      /* x+x may overflow */
 57                         ss = s-c;
 58                         cc = s+c;
119                 } else if (signbitl(s) != signbitl(c)) {
120                         ss = s - c;
121                         cc = -cosl(x + x) / ss;
 61                         cc = -cosl(x+x)/ss;
122                 } else {
123                         cc = s + c;
124                         ss = -cosl(x + x) / cc;
 64                         ss = -cosl(x+x)/cc;
125                 }

127 #endif /* ! codereview */
128                 /*
129                  * j0(x) = 1/sqrt(pi) * (P(0,x)*cc - Q(0,x)*ss) / sqrt(x)
130                  * y0(x) = 1/sqrt(pi) * (P(0,x)*ss + Q(0,x)*cc) / sqrt(x)
131                  */
132                 if (x > 1.0e120L)
133                         return ((invsqrtpi * cc) / sqrtl(x));

135                 u = pzero(x);
136                 v = qzero(x);
137                 return (invsqrtpi * (u * cc - v * ss) / sqrtl(x));
138         }

140         if (x <= small) {
141                 if (x <= tiny)
142                         return (one - x);
143                 else
144                         return (one - x * x * 0.25L);
 66                 if (x>1.0e120L) return (invsqrtpi*cc)/sqrtl(x);
 67                 u = pzero(x); v = qzero(x);
 68                 return invsqrtpi*(u*cc-v*ss)/sqrtl(x);
 69         }
 70         if (x<=small) {
 71             if (x<=tiny) return one-x;
 72             else return one-x*x*0.25L;
 73         }
 74         z = x*x;
 75         r = r0[6]; s = s0[6];
 76         for(i=5;i>=0;i--) {
 77             r = r*z + r0[i];
 78             s = s*z + s0[i];
145         }

147         z = x * x;
148         r = r0[6];
149         s = s0[6];

151         for (i = 5; i >= 0; i--) {
152                 r = r * z + r0[i];
153                 s = s * z + s0[i];
154         }

156         return (one + z * (r / s));
 80         return(one+z*(r/s));
157 }
```

_____*unchanged_portion_omitted_*

```
170 #endif /* ! codereview */
171 static const GENERIC v0[8] = {
```



```
172         1.0e0L,
173         1.2471644165391113115716767661277671279 70e-0002L,
174         7.8291447496397915000529002814893674435 76e-0005L,
175         3.2471265404222453305112183210133603366 06e-0007L,
176         9.7505167247894996785670625725495684478 69e-0010L,
177         2.1567132231735912122505433902584580987 76e-0012L,
178         3.3221695615978900042314824312364527526 24e-0015L,
179         2.8212132953140009242522264863057268050 93e-0018L,
180 };

182 GENERIC
183 y0l(GENERIC x)
184 {
185         GENERIC z, s, c, ss, cc, u, v;
 93 y0l(x) GENERIC x;{
 94         GENERIC z, s,c,ss,cc,u,v;
186         int i;
187         volatile GENERIC d;

189         if (isnanl(x))
190                 return (x + x);

 98         if (isnanl(x)) return x+x;
192         if (x <= zero) {
193                 if (x == zero)
194                         d = -one / (x - x);
101                     d= -one/(x-x);
195                 else
196                         d = zero / (x - x);
103                     d = zero/(x-x);
197         }

199 #endif /* ! codereview */
200 #ifdef lint
201         d = d;
202 #endif

204 #endif /* ! codereview */
205         if (x > 1.28L) {
206                 if (!finitel(x))
207                         return (zero);

105                 if (!finitel(x)) return zero;
209                 s = sinl(x);
210                 c = cosl(x);

212                 /* BEGIN CSTYLED */
213 #endif /* ! codereview */
214                 /*
215                  * j0(x) = sqrt(2/(pi*x))*(p0(x)*cos(x0)-q0(x)*sin(x0))
216                  * where x0 = x-pi/4
217                  *      Better formula:
218                  *              cos(x0) = cos(x)cos(pi/4)+sin(x)sin(pi/4)
219                  *                      = 1/sqrt(2) * (cos(x) + sin(x))
220                  *              sin(x0) = sin(x)cos(pi/4)-cos(x)sin(pi/4)
221                  *                      = 1/sqrt(2) * (sin(x) - cos(x))
222                  * To avoid cancellation, use
223                  *              sin(x) +- cos(x) = -cos(2x)/(sin(x) -+ cos(x))
224                  * to compute the worse one.
225                  */
226                 /* END CSTYLED */
227                 if (x > 1.0e2450L) {    /* x+x may overflow */
228                         ss = s - c;
229                         cc = s + c;
108                 if (x>1.0e2450L) {      /* x+x may overflow */
109                         ss = s-c;
```

```
110                         cc = s+c;
230                 } else if (signbitl(s) != signbitl(c)) {
231                         ss = s - c;
232                         cc = -cosl(x + x) / ss;
113                         cc = -cosl(x+x)/ss;
233                 } else {
234                         cc = s + c;
235                         ss = -cosl(x + x) / cc;
116                         ss = -cosl(x+x)/cc;
236                 }

238 #endif /* ! codereview */
239                 /*
240                  * j0(x) = 1/sqrt(pi*x) * (P(0,x)*cc - Q(0,x)*ss)
241                  * y0(x) = 1/sqrt(pi*x) * (P(0,x)*ss + Q(0,x)*cc)
242                  */
243                 if (x > 1.0e120L)
244                         return ((invsqrtpi * ss) / sqrtl(x));
118                 if (x>1.0e120L) return (invsqrtpi*ss)/sqrtl(x);
119                 return invsqrtpi*(pzero(x)*ss+qzero(x)*cc)/sqrtl(x);

246                 return (invsqrtpi * (pzero(x) * ss + qzero(x) * cc) / sqrtl(x));
247 #endif /* ! codereview */
248         }

250         if (x <= tiny)
251                 return (u0[0] + tpi * logl(x));

253         z = x * x;
254         u = u0[7];
255         v = v0[7];

257         for (i = 6; i >= 0; i--) {
258                 u = u * z + u0[i];
259                 v = v * z + v0[i];
121         if (x<=tiny) {
122             return (u0[0] + tpi*logl(x));
260         }

262         return (u / v + tpi * (j0l(x) * logl(x)));
124         z = x*x;
125         u = u0[7]; v = v0[7];
126         for(i=6;i>=0;i--) {
127             u = u*z + u0[i];
128             v = v*z + v0[i];
129         }
130         return(u/v + tpi*(j0l(x)*logl(x)));
263 }
_____unchanged_portion_omitted_

280 #endif /* ! codereview */
281 static const GENERIC ps0[11] = {
282         1.0e0L,
283         1.06605192787727337642536582396755051268 7e+0003L,
284         4.39173964716838159239917380432926635303 8e+0005L,
285         9.07516226180134367180565829412388886788 4e+0007L,
286         1.02318611851990475181958191207598599505 8e+0010L,
287         6.42786186041422374634051537651273027506 1e+0011L,
288         2.21086150323782358973548130362799340623 5e+0013L,
289         3.94324733578429290591595684090181817798 9e+0014L,
290         3.28372097677754514215020011064727000448 1e+0015L,
291         1.04534691881275404890364564153872898675 9e+0016L,
292         8.04345546806561890075059958429119368046 3e+0015L,
293 };

295 #endif /* ! codereview */
```



```
296 static const GENERIC pr1[12] = {         /* [8 -- 16] */
297         9.999999999999999999997844227011086836 18e-0001L,
298         6.796098532948334207755488692777907062894e+0002L,
299         1.840036112605722168824530758797169836042e+0005L,
300         2.598490483191916637264894340635847598122e+0007L,
301         2.105774863242707025525730249472054578523e+0009L,
302         1.015822044230542426666314997796944979959e+0011L,
303         2.931557457008110436764077699944189071875e+0012L,
304         4.962885121125457633655259224179322808824e+0013L,
305         4.705424055148223269155430598563351566279e+0014L,
306         2.294439854910747229152056080910427001110e+0015L,
307         4.905531843137486691500950019322475458629e+0015L,
308         3.187543169710339218793442542845735994565e+0015L,
309 };

311 #endif /* ! codereview */
312 static const GENERIC ps1[14] = {
313         1.0e0L,
314         6.796801657948334207754571576066758180288e+0002L,
315         1.840512891201300567325421059826676366447e+0005L,
316         2.599777028312918975306252167127695075221e+0007L,
317         2.107582572771047636846811284634244892537e+0009L,
318         1.017275794694156108975782763889979940348e+0011L,
319         2.938487645192463845428059755454762316011e+0012L,
320         4.982512164735557054521042916182317924466e+0013L,
321         4.737639900153703274792677468264564361437e+0014L,
322         2.323398719123742743524249528275097100646e+0015L,
323         5.033419107069210577868909797896984419391e+0015L,
324         3.409036105931068609601317076759804716059e+0015L,
325         7.505655364352679737585745147753521662166e+0013L,
326         -9.976837153983688250780198248297109118313e+0012L,
327 };

329 #endif /* ! codereview */
330 static const GENERIC pr2[12] = {         /* [5 -- 8 ] */
331         9.999999999999999937857236789277366320220e-0001L,
332         3.692848765268649571651602420376358849214e+0002L,
333         5.373022067535476576926715900057760985410e+0004L,
334         4.038738891191314969971504035057219430725e+0006L,
335         1.728285706306940523397385566659762646999e+0008L,
336         4.375400819645889911158688737206054788534e+0009L,
337         6.598950418204912408375591217782088567076e+0010L,
338         5.827182039183238492480275401520072793783e+0011L,
339         2.884222642913492390887572414999490975844e+0012L,
340         7.373278873797767721932837830628688632775e+0012L,
341         8.338295457568973761205077964397969230489e+0012L,
342         2.911383183467288345772308817209806922143e+0012L,
343 };

345 #endif /* ! codereview */
346 static const GENERIC ps2[14] = {
347         1.0e0L,
348         3.693551890268649477288896267171993213102e+0002L,
349         5.375607880998361502474715133828068514297e+0004L,
350         4.042477764024108249744998862572786367328e+0006L,
351         1.731069838737016956685839588670132939513e+0008L,
352         4.387147674049898778738226585935491417728e+0009L,
353         6.628058659620653765349556940567715258165e+0010L,
354         5.869659904164177740471685856367322160664e+0011L,
355         2.919839445622817017058977559638969436383e+0012L,
356         7.535314897696671402628203718612309253907e+0012L,
357         8.696355561452933775733098597486106589 35e+0012L,
358         3.216155103141537221173601557697083216257e+0012L,
359         4.756857081068942248246880159213789086363e+0010L,
360         -3.496356619666608032231074866481472824067e+0009L,
361 };
```

```
#endif /* ! codereview */
static const GENERIC pr3[13] = {        /* [3.5 -- 5 ] */
        9.999999999999916693107285612398196588247e-0001L,
        2.263975921282917721194425320484974336945e+0002L,
        1.994358386744245848889492762781484199966e+0004L,
        8.980067458430542243559962493831661323168e+0005L,
        2.282213787521372663705567756420087553508e+0007L,
        3.409784374889063618250288699908375135923e+0008L,
        3.024380857401448589254343517589811711108e+0009L,
        1.571110368046740246895071721443082286379e+0010L,
        4.603187020243604632153685300463160593768e+0010L,
        7.087196453409712719449549280664058793403e+0010L,
        5.046196021776346356803687409644239065041e+0010L,
        1.287758439080165765709154276618854799932e+0010L,
        5.900679773415023433787846658096813590784e+0008L,
};

#endif /* ! codereview */
static const GENERIC ps3[13] = {
        1.0e0L,
        2.264679046282855061328604619231774747116e+0002L,
        1.995939523988944553755653255389812103448e+0004L,
        8.993853144706348727038389967490183236820e+0005L,
        2.288326099634588843906989983704795468773e+0007L,
        3.424967100255240885169240956804790118282e+0008L,
        3.046311797972463991368023759640028910016e+0009L,
        1.589614961932826812790222479700797224003e+0010L,
        4.692406624527744816497089139325073939927e+0010L,
        7.320486495902008912866462849073108323948e+0010L,
        5.345945972828978289935309597742981360994e+0010L,
        1.444033091910423754121309915092247171008e+0010L,
        7.987714685115314668378957273824383610525e+0008L,
};

#endif /* ! codereview */
static const GENERIC pr4[13] = {        /* [2.5, 3.5] */
        9.999999999867366779611187227477577122 60e-0001L,
        1.453824980703800559037873123568378845663e+0002L,
        8.097327216430682288267610447006508661032e+0003L,
        2.273847252038264370231169686380192662135e+0005L,
        3.561056728046211111354759998976985449622e+0006L,
        3.244933588800096378434627029369680378599e+0007L,
        1.740112392860717950376210038908476792588e+0008L,
        5.426170187455893285197878563881579269524e+0008L,
        9.490107486454362321004377336020526281371e+0008L,
        8.688872439428470049801714121070005313806e+0008L,
        3.673315853166437222811910656900123215515e+0008L,
        5.577770470359033051648774463396932703 9e+0007L,
        1.540438642031689641308197880181291865714e+0006L,
};

#endif /* ! codereview */
static const GENERIC ps4[13] = {        /* [2.5, 3.5] */
        1.0e0L,
        1.454528105698159439773035951959131799816e+0002L,
        8.107442215200392397172179900434987859618e+0003L,
        2.279390393778242887574177096606328994140e+0005L,
        3.576251625592252008424781111770934135844e+0006L,
        3.267909499056932631405942058670933813863e+0007L,
        1.760021515330805537499778238099704648805e+0008L,
        5.525553787667353981242060222587465726729e+0008L,
        9.769870295912820457889384082671269328511e+0008L,
        9.110582071004774279226905629401808454e+0008L,
        3.981857678621955599371967680343918454e+0008L,
        6.482404686230769399073192961667697036706e+0007L,
```



```
        2.210046943095878402443535460329391782298e+0006L,
};

#endif /* ! codereview */
static const GENERIC pr5[13] = {        /* [1.777..., 2.5] */
        9.999999999114986107951817871144655880699e-0001L,
        9.252583736048588342568344570315435947614e+0001L,
        3.218726757856078715214631502407386264637e+0003L,
        5.554009964621111656479588505862577040831e+0004L,
        5.269993115643664338253196944523510290175e+0005L,
        2.874613773778430691192912190618220544575e+0006L,
        9.133538151103658353874146919613442436035e+0006L,
        1.673067041410338922825193013077354249193e+0007L,
        1.706913873848398011744790289200151840498e+0007L,
        9.067766583853288534551600235576747618679e+0006L,
        2.216746733457884568532695355036338655872e+0006L,
        1.945753880802872541235703812722344514405e+0005L,
        3.132374412921948071539195638885330951749e+0003L,
};

#endif /* ! codereview */
static const GENERIC ps5[13] = {        /* [1.777..., 2.5] */
        1.0e0L,
        9.259614983862181118883831670990340052982e+0001L,
        3.225125275462903384842124075132609290304e+0003L,
        5.575705362829101545292760055941855246492e+0004L,
        5.306049863037087855496170121958448492522e+0005L,
        2.907060758873509564309729903109018597215e+0006L,
        9.298059206584995898298257827131208539289e+0006L,
        1.720391071006963176836108026556547062980e+0007L,
        1.782614812922865190479394509487941920612e+0007L,
        9.708016389605273153536452032839879950155e+0006L,
        2.476495084688170096480215640962175140027e+0006L,
        2.363200660365585759668077790194604917187e+0005L,
        4.803239569848196077121203575704356936731e+0003L,
};

#endif /* ! codereview */
static const GENERIC pr6[13] = {        /* [1.28, 1.777...] */
        9.999999969777095495998606925524322559556e-0001L,
        5.825486719466194430503283824096872219216e+0001L,
        1.248155491637757281915184824965379905380e+0003L,
        1.302093199842358609321338417071710477615e+0004L,
        7.353835804186292782840961999810543016039e+0004L,
        2.356471661113686180549195092555751341757e+0005L,
        4.350553267429009581632987060942780847101e+0005L,
        4.588762661876600638719159826652389418235e+0005L,
        2.675796398548523436544221045225290128611e+0005L,
        8.077649557108971388298292919988449940464e+0004L,
        1.117640459221306873519068741664054573776e+0004L,
        5.544400072396814695175787511557757885585e+0002L,
        5.072550541191480498431289089905822910718e+0000L,
};

#endif /* ! codereview */
static const GENERIC ps6[13] = {        /* [1.28, 1.777...] */
        1.0e0L,
        5.832517925357165050639075848183613063291e+0001L,
        1.252144364743592128171256104364976466898e+0003L,
        1.310300234342216813579118022415585740772e+0004L,
        7.434667697093812197817292154032863632923e+0004L,
        2.398706595587719165726469002404004614711e+0005L,
        4.472737517625103157004869372427480602511e+0005L,
        4.786313523337761975294171429067037236 11e+0005L,
        2.851161872872731228472536061865365370192e+0005L,
        8.891648269899148412331918021801385815586e+0004L,
```

```
494         1.297097489535351517572978123584751042287e+0004L,
495         7.096761640545975756202184143400469812618e+0002L,
496         8.378049338590233325977702401733340820351e+0000L,
497 };

499 #endif /* ! codereview */
500 static const GENERIC sixteen = 16.0L;
501 static const GENERIC huge = 1.0e30L;

503 static GENERIC
504 pzero(GENERIC x)
147 static GENERIC pzero(x)
148 GENERIC x;
505 {
506         GENERIC s, r, t, z;
150         GENERIC s,r,t,z;
507         int i;

509         if (x > huge)
510                 return (one);

512         t = one / x;
513         z = t * t;

515         if (x > sixteen) {
516                 r = z * pr0[11] + pr0[10];
517                 s = ps0[10];

519                 for (i = 9; i >= 0; i--) {
520                         r = z * r + pr0[i];
521                         s = z * s + ps0[i];
152         if (x>huge) return one;
153         t = one/x; z = t*t;
154         if (x>sixteen) {
155             r = z*pr0[11]+pr0[10]; s = ps0[10];
156             for (i=9;i>=0;i--) {
157                 r = z*r + pr0[i];
158                 s = z*s + ps0[i];
522                 }
523         } else if (x > eight) {
524                 r = pr1[11];
525                 s = ps1[11] + z * (ps1[12] + z * ps1[13]);

527                 for (i = 10; i >= 0; i--) {
528                         r = z * r + pr1[i];
529                         s = z * s + ps1[i];
161             r = pr1[11]; s = ps1[11]+z*(ps1[12]+z*ps1[13]);
162             for (i=10;i>=0;i--) {
163                 r = z*r + pr1[i];
164                 s = z*s + ps1[i];
530                 }
531         } else if (x > five) {          /* x > 5.0 */
532                 r = pr2[11];
533                 s = ps2[11] + z * (ps2[12] + z * ps2[13]);

535                 for (i = 10; i >= 0; i--) {
536                         r = z * r + pr2[i];
537                         s = z * s + ps2[i];
538                 }
539         } else if (x > 3.5L) {
540                 r = pr3[12];
541                 s = ps3[12];

543                 for (i = 11; i >= 0; i--) {
544                         r = z * r + pr3[i];
545                         s = z * s + ps3[i];
```



```
546                 }
547         } else if (x > 2.5L) {
548                 r = pr4[12];
549                 s = ps4[12];

551                 for (i = 11; i >= 0; i--) {
552                         r = z * r + pr4[i];
553                         s = z * s + ps4[i];
554                 }
555         } else if (x > (1.0L / 0.5625L)) {
556                 r = pr5[12];
557                 s = ps5[12];

559                 for (i = 11; i >= 0; i--) {
560                         r = z * r + pr5[i];
561                         s = z * s + ps5[i];
167             r = pr2[11]; s = ps2[11]+z*(ps2[12]+z*ps2[13]);
168             for (i=10;i>=0;i--) {
169                 r = z*r + pr2[i];
170                 s = z*s + ps2[i];
171             }
172         } else if (x>3.5L) {
173             r = pr3[12]; s = ps3[12];
174             for (i=11;i>=0;i--) {
175                 r = z*r + pr3[i];
176                 s = z*s + ps3[i];
177             }
178         } else if (x>2.5L) {
179             r = pr4[12]; s = ps4[12];
180             for (i=11;i>=0;i--) {
181                 r = z*r + pr4[i];
182                 s = z*s + ps4[i];
183             }
184         } else if (x> (1.0L/0.5625L)) {
185             r = pr5[12]; s = ps5[12];
186             for (i=11;i>=0;i--) {
187                 r = z*r + pr5[i];
188                 s = z*s + ps5[i];
562                 }
563         } else {                        /* assume x > 1.28 */
564                 r = pr6[12];
565                 s = ps6[12];

567                 for (i = 11; i >= 0; i--) {
568                         r = z * r + pr6[i];
569                         s = z * s + ps6[i];
191             r = pr6[12]; s = ps6[12];
192             for (i=11;i>=0;i--) {
193                 r = z*r + pr6[i];
194                 s = z*s + ps6[i];
570             }
571         }

573         return (r / s);
197         return r/s;
574 }

575 static const GENERIC qr0[12] = {         /* [16, inf] */
576         -1.249999999999999999999999999999999972972e-0001L,
577         -1.425179595545670577414395762503991596897e+0002L,
578         -6.312499645625970845534460257936222407219e+0004L,
579         -1.411374326457208384315121243698814446848e+0007L,
580         -1.735034212758873581410984757860787252842e+0009L,
581         -1.199777647512789489421826342485055280680e+0011L,
582         -4.596025334081655714499860409699100373644e+0012L,
```

```
        -9.2625256282012841077929244770316533991 87e+0013L,
        -8.8583947286850392453443988421806628676 39e+0014L,
        -3.2675279536875348876237406227095059721 13e+0015L,
        -2.6642229711863119675871293470294500620 19e+0015L,
        3.4424640607239878695851800953445041002 04e+0014L,
};

#endif /* ! codereview */
static const GENERIC qs0[11] = {
        1.0e0L,
        1.1407296139365364619315166100031856878 81e+0003L,
        5.0566655104422993510091981864900858035 80e+0005L,
        1.1320417638256427879439416505227181991 15e+0008L,
        1.3945701118725816063926206782142464797 67e+0010L,
        9.6779452181522647895344310795637443784 21e+0011L,
        3.7311403278515368282251430588963485020 96e+0013L,
        7.6127859510648692917228466810208816764 10e+0014L,
        7.4760770164067648917301910048118639759 40e+0015L,
        2.9512464826135920354215034271003938317 09e+0016L,
        3.1083618036918117111368545870743020349 01e+0016L,
};

#endif /* ! codereview */
static const GENERIC qr1[12] = {        /* [8, 16 ] */
        -1.2499999999999999999979490103834338185 7e-0001L,
        -9.0512151663938226406367522448951241269 34e+0001L,
        -2.6207827034281488376711790319042083039 47e+0004L,
        -3.9755712615535044577661779745087859088 4e+0006L,
        -3.4790293307593113062700722187407497409 0e+0008L,
        -1.8239550081242685730362167461862398290 89e+0010L,
        -5.7659326971118013757651560292215686644 35e+0011L,
        -1.0798436807987425929540021924179347791 14e+0013L,
        -1.1468936305045927390822057646115813328 97e+0014L,
        -6.3670160596838984649361044472828807041 82e+0014L,
        -1.5831090419612134904644591119034842090 98e+0015L,
        -1.2301495557642424731031286501357956394 12e+0015L,
};

#endif /* ! codereview */
static const GENERIC qs1[14] = {
        1.0e0L,
        7.2468315081150581124385798477780144584 32e+0002L,
        2.1008541844391685183993837863069270376 11e+0005L,
        3.1926364188379515074301882859409942351 22e+0007L,
        2.8015584433833546745384434611244342161 52e+0009L,
        1.4750269976643737392934839272506536748 7e+0011L,
        4.6944868249139546085523638217999271453 18e+0012L,
        8.8903501009192002508384387096015473340 21e+0013L,
        9.6268444290829051448747010687604697520 67e+0014L,
        5.5411107446004607735282638626875216421 40e+0015L,
        1.4865004947894525567274703292321230965 63e+0016L,
        1.4158401048459594003654307737320938992 10e+0016L,
        1.7808660952415174180813125672396823364 83e+0015L,
        -2.3592309173848893578876315440799901294 94e+0014L,
};

#endif /* ! codereview */
static const GENERIC qr2[12] = {        /* [5, 8] */
        -1.2499999999999995319377443625277721816 14e-0001L,
        -4.9443738973569697748393759772392415739 66e+0001L,
        -7.7284491754334652853142616500784504739 09e+0003L,
        -6.2625743296127523463369014346512207059 03e+0005L,
        -2.9009482202209433060272352174243806727 32e+0007L,
        -7.9887196476341927704639171575628741195 35e+0008L,
        -1.3182281719271813895477600266263570123 75e+0010L,
        -1.2824397739830292453092632719454242819 6e+0011L,
```



```
        -7.0509255708278180401861499402579188451 38e+0011L,
        -2.0217518825738719900042056187420268442 9e+0012L,
        -2.5929399624006685523843339005738126356 58e+0012L,
        -1.0382671095188912628406015149329728503 26e+0012L,
};

#endif /* ! codereview */
static const GENERIC qs2[14] = {
        1.0e0L,
        3.9613584928855700032027840228942489521 16e+0002L,
        6.2057887388647018828287526345865109269 68e+0004L,
        5.0457156039326702865506738130117644067 49e+0006L,
        2.3492486113626583233533433894309687514 29e+0008L,
        6.5202445244158286359176835537218800639 11e+0009L,
        1.0891112112235077193370671598862818877 22e+0011L,
        1.0804060009053598679587794094149030186 10e+0012L,
        6.1356452808955147035141546806237695621 48e+0012L,
        1.8624330402466258742458671513686436682 15e+0013L,
        2.6677808057866488888407778887021937089 94e+0013L,
        1.3944011072890877747653007118093131128 24e+0013L,
        1.0932475006163203755628982971567224454 84e+0012L,
        -7.2288755303789287228266042164914937807 75e+0010L,
};

#endif /* ! codereview */
static const GENERIC qr3[13] = {        /* [3.5 5] */
        -1.2499999999994730677484203795784816610 75e-0001L,
        -3.0445490486352893519135743248032509779 98e+0001L,
        -2.8900811406497690784966930035246814408 69e+0003L,
        -1.4049224568172022358793432753305291076 84e+0005L,
        -3.8627466143855734435181774036173492818 69e+0006L,
        -6.2575173091102490492011337089111550476 89e+0007L,
        -6.0314513309208399169870797827273234775 20e+0008L,
        -3.4115424051738306114540257657558543823 46e+0009L,
        -1.0893924781497266721330144987230215260 99e+0010L,
        -1.8249340784202109412901409034159567827 26e+0010L,
        -1.4007802783043587104234810704869395311 39e+0010L,
        -3.7164841360649173639266357167437710920 93e+0009L,
        -1.3975910752964255299704348909549043315 80e+0008L,
};

#endif /* ! codereview */
static const GENERIC qs3[13] = {
        1.0e0L,
        2.4414986139049620493910001870149458580 42e+0002L,
        2.3261888820723707115001642223415143370 43e+0004L,
        1.1371382131212313384949771046592395781 65e+0006L,
        3.1529180707356627287229984526053642535 17e+0007L,
        5.1728779934265072593142704884440135951 08e+0008L,
        5.0830864397316698074559610788564707741 15e+0009L,
        2.9618427320664341231193255211394769099 41e+0010L,
        9.9121858668624407358297818560813531513 90e+0010L,
        1.7935605612516222344305641815672979835 98e+0011L,
        1.5770901193412281225252651084979404030 73e+0011L,
        5.5099103067801661943338899999854636816 36e+0010L,
        4.7616911340788744912023201815179367581 41e+0009L,
};

#endif /* ! codereview */
static const GENERIC qr4[13] = {        /* [2.5 3.5] */
        -1.2499999999285677343397450434907053408 35e-0001L,
        -1.9672017487314190630516016244355655284 81e+0001L,
        -1.1863291467145622364070997406155281707 07e+0003L,
        -3.6077369592229418103563014911524579340 60e+0004L,
        -6.1192007179781049049328284685751942671 25e+0005L,
        -6.0378477811583582266703050786522055863 84e+0006L,
```

```
        -3.503558153336140359700536720393565984740e+0007L,
        -1.180196478268225718757218523746787309773e+0008L,
        -2.221860232085134915841426363505169680528e+0008L,
        -2.173372505452747585296176761701746236760e+0008L,
        -9.649364865061237558517730539506568013963e+0007L,
        -1.465429227847933034546039640094862650385e+0007L,
        -3.083003197920262085170581866246663380607e+0005L,
};

#endif /* ! codereview */
static const GENERIC qs4[13] = {        /* [2.5 3.5] */
        1.0e0L,
        1.579620773732259142752614142139986854055e+0002L,
        9.581372220329138733203879503753685054968e+0003L,
        2.939598672379108095776114131010825885308e+0005L,
        5.052183049314542218630341818692588448168e+0006L,
        5.083497695595206639433839326338971980149e+0007L,
        3.036385361800553388049719014005099206516e+0008L,
        1.067826481452753409910563785161661492137e+0009L,
        2.145644125557118044720741775125319669272e+0009L,
        2.324115615959719949363946673491552216799e+0009L,
        1.223262962112070757966959855619847011146e+0009L,
        2.569765553318495423738478585947110270709e+0008L,
        1.354744744299227127897905787732636565504e+0007L,
};

#endif /* ! codereview */
static const GENERIC qr5[13] = {        /* [1.777.., 2.5] */
        -1.249999995936639697637680428174576069971e-0001L,
        -1.260846055371311453485891923426489068315e+0001L,
        -4.772398467544467480801174330290141578895e+0002L,
        -8.939852599990298486613760833996490599724e+0003L,
        -9.184070787149542050979542226446134243197e+0004L,
        -5.406038945018274458362637897739280435171e+0005L,
        -1.845896544705190261018653728678171084418e+0006L,
        -3.613616990680809501878667570653308071547e+0006L,
        -3.908782978135693252252557720414348623779e+0006L,
        -2.173711022517323927109138170588442768176e+0006L,
        -5.431253130679918485836408549007856244495e+0005L,
        -4.591098546452684510082591587275940765959e+0004L,
        -5.244711364168207806835520057792229646578e+0002L,
};

#endif /* ! codereview */
static const GENERIC qs5[13] = {        /* [1.777.., 2.5] */
        1.0e0L,
        1.014536210851290878350892750972474861447e+0002L,
        3.875547510687135314064434160096139681076e+0003L,
        7.361913122670079814955259281995617732580e+0004L,
        7.720288944218771126581086539585529314636e+0005L,
        4.681529554446752496404431433608306558038e+0006L,
        1.667882621940503925455031252308367745820e+0007L,
        3.469403153761399881888272620855305156241e+0007L,
        4.096992047964210711867089384719947863019e+0007L,
        2.596804755829217449311530735959560630554e+0007L,
        7.983933774697889238154465064019410763845e+0006L,
        9.818133816979900819087242425280757938152e+0005L,
        3.061083930868694396013541535670745443560e+0004L,
};

static const GENERIC qr6[13] = {        /* [1.28, 1.777..] */
        -1.249999881577289001807137282824929082771e-0001L,
        -7.998273510053110759610810594119533619282e+0000L,
        -1.872481955335172543369089617771565632719e+0002L,
        -2.122116786726308050798740033037996468 12e+0003L,
        -1.293850285839529282503178263484773478457e+0004L,
```



```
        -4.445024742266316181033354192262529356093e+0004L,
        -8.730161378334357767668344467356505347070e+0004L,
        -9.706222895172078442801444972505315054736e+0004L,
        -5.896325518259858270165531513618195321041e+0004L,
        -1.823172034368108822276420827074668832233e+0004L,
        -2.509304178635055926638833040337472387175e+0003L,
        -1.156608965715779237316769828941729964099e+0002L,
        -7.028005789650731396887346826397785210442e-0001L,
};

#endif /* ! codereview */
static const GENERIC qs6[13] = {        /* [1.28, 1.777..] */
        1.0e0L,
        6.457211085058064845601261321277721075900e+0001L,
        1.534005216588011210342824555136008682950e+0003L,
        1.777217999176441782593357660462379097171e+0004L,
        1.118372652642469468091084810263231199696e+0005L,
        4.015242433858461813142365748386473605294e+0005L,
        8.377081045517098645448616514388280497673e+0005L,
        1.011495020008010352575398009604164287337e+0006L,
        6.886722075290430568652227875200208955970e+0005L,
        2.504735189948021472047157148613171956537e+0005L,
        4.408138920171044846941001844352009817062e+0004L,
        3.105572178072115145673058722853640854884e+0003L,
        5.588294821118916113437396504573817033678e+0001L,
};

static GENERIC
qzero(GENERIC x)
static GENERIC qzero(x)
GENERIC x;
{
        GENERIC s, r, t, z;
        GENERIC s,r,t,z;
        int i;

        if (x > huge)
                return (-0.125L / x);

        t = one / x;
        z = t * t;

        if (x > sixteen) {
                r = z * qr0[11] + qr0[10];
                s = qs0[10];

                for (i = 9; i >= 0; i--) {
                        r = z * r + qr0[i];
                        s = z * s + qs0[i];
                }
        } else if (x > eight) {
                r = qr1[11];
                s = qs1[11] + z * (qs1[12] + z * qs1[13]);

                for (i = 10; i >= 0; i--) {
                        r = z * r + qr1[i];
                        s = z * s + qs1[i];
                }
        } else if (x > five) {  /* assume x > 5.0 */
                r = qr2[11];
                s = qs2[11] + z * (qs2[12] + z * qs2[13]);

                for (i = 10; i >= 0; i--) {
                        r = z * r + qr2[i];
                        s = z * s + qs2[i];
                }
```

```
        } else if (x > 3.5L) {
                r = qr3[12];
                s = qs3[12];

                for (i = 11; i >= 0; i--) {
                        r = z * r + qr3[i];
                        s = z * s + qs3[i];
                }
        } else if (x > 2.5L) {
                r = qr4[12];
                s = qs4[12];

                for (i = 11; i >= 0; i--) {
                        r = z * r + qr4[i];
                        s = z * s + qs4[i];
                }
        } else if (x > (1.0L / 0.5625L)) {
                r = qr5[12];
                s = qs5[12];

                for (i = 11; i >= 0; i--) {
                        r = z * r + qr5[i];
                        s = z * s + qs5[i];
        if (x>huge) return -0.125L/x;
        t = one/x; z = t*t;
        if (x>sixteen) {
            r = z*qr0[11]+qr0[10]; s = qs0[10];
            for (i=9;i>=0;i--) {
                r = z*r + qr0[i];
                s = z*s + qs0[i];
            }
        } else if (x>eight) {
            r = qr1[11]; s = qs1[11]+z*(qs1[12]+z*qs1[13]);
            for (i=10;i>=0;i--) {
                r = z*r + qr1[i];
                s = z*s + qs1[i];
            }
        } else if (x>five) {  /* assume x > 5.0 */
            r = qr2[11]; s = qs2[11]+z*(qs2[12]+z*qs2[13]);
            for (i=10;i>=0;i--) {
                r = z*r + qr2[i];
                s = z*s + qs2[i];
            }
        } else if (x>3.5L) {
            r = qr3[12]; s = qs3[12];
            for (i=11;i>=0;i--) {
                r = z*r + qr3[i];
                s = z*s + qs3[i];
            }
        } else if (x>2.5L) {
            r = qr4[12]; s = qs4[12];
            for (i=11;i>=0;i--) {
                r = z*r + qr4[i];
                s = z*s + qs4[i];
            }
        } else if (x> (1.0L/0.5625L)) {
            r = qr5[12]; s = qs5[12];
            for (i=11;i>=0;i--) {
                r = z*r + qr5[i];
                s = z*s + qs5[i];
                }
        } else {                        /* assume x > 1.28 */
                r = qr6[12];
                s = qs6[12];

                for (i = 11; i >= 0; i--) {
```



```
                        r = z * r + qr6[i];
                        s = z * s + qs6[i];
            r = qr6[12]; s = qs6[12];
            for (i=11;i>=0;i--) {
                r = z*r + qr6[i];
                s = z*s + qs6[i];
                }
        }

        return (t * (r / s));
        return t*(r/s);
}
```

_____*unchanged_portion_omitted_*

```
**********************************************************
   27342 Thu Jun  6 20:38:47 2019
new/usr/src/lib/libm/common/Q/j1l.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
   1 /*
   2  * CDDL HEADER START
   3  *
   4  * The contents of this file are subject to the terms of the
   5  * Common Development and Distribution License (the "License").
   6  * You may not use this file except in compliance with the License.
   7  *
   8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
   9  * or http://www.opensolaris.org/os/licensing.
  10  * See the License for the specific language governing permissions
  11  * and limitations under the License.
  12  *
  13  * When distributing Covered Code, include this CDDL HEADER in each
  14  * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  15  * If applicable, add the following below this CDDL HEADER, with the
  16  * fields enclosed by brackets "[]" replaced with your own identifying
  17  * information: Portions Copyright [yyyy] [name of copyright owner]
  18  *
  19  * CDDL HEADER END
  20  */

  22 /*
  23  * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
  24  */

  26 #endif /* ! codereview */
  27 /*
  28  * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
  29  * Use is subject to license terms.
  30  */

  32 /*
  33  * floating point Bessel's function of the first and second kinds
  34  * of order zero: j1(x),y1(x);
  35  *
  36  * Special cases:
  37  *      y0(0)=y1(0)=yn(n,0) = -inf with division by zero signal;
  38  *      y0(-ve)=y1(-ve)=yn(n,-ve) are NaN with invalid signal.
  39  */

  41 #pragma weak __j1l = j1l
  42 #pragma weak __y1l = y1l

  44 #include "libm.h"
  45 #include "longdouble.h"

  47 #define GENERIC long double

  49 static const GENERIC zero = 0.0L,
  50         small = 1.0e-9L,
  51         tiny = 1.0e-38L,
  52         one = 1.0L,
  53         five = 5.0L,
  54         invsqrtpi = 5.641895835477562869480794515607725858441e-0001L,
  55         tpi = 0.636619772367581343075535053490057448L;
  25 static const GENERIC
  26 zero    = 0.0L,
  27 small   = 1.0e-9L,
  28 tiny    = 1.0e-38L,
  29 one     = 1.0L,
  30 five    = 5.0L,
```



```
  31 invsqrtpi= 5.641895835477562869480794515607725858441e-0001L,
  32 tpi     = 0.636619772367581343075535053490057448L;

  57 static GENERIC pone(GENERIC);
  58 static GENERIC qone(GENERIC);

  60 static const GENERIC r0[7] = {
  61         -6.249999999999999999999999999999999627320e-0002L,
  62         1.940606727194041716205384618494641565464e-0003L,
  63         -3.005630423155733701856481469986459043883e-0005L,
  64         2.345586219403918667468341047369572169358e-0007L,
  65         -9.976809285885253587529010109133336669724e-0010L,
  66         2.218743258363623946078958783775107473381e-0012L,
  67         -2.071079656218700604767650924103578046280e-0015L,
  68 };

  70 #endif /* ! codereview */
  71 static const GENERIC s0[7] = {
  72         1.0e0L,
  73         1.061695903156199920738051277075003059555e-0002L,
  74         5.521860513111180371566951179398862692060e-0005L,
  75         1.824214367413754193524107877084979441407e-0007L,
  76         4.098957778439576834818838198039029353925e-0010L,
  77         6.047735079699666389853240090925264056197e-0013L,
  78         4.679044728878836197247923279512047035041e-0016L,
  79 };

  81 GENERIC
  82 j1l(GENERIC x)
  83 {
  84         GENERIC z, d, s, c, ss, cc, r;
  46 j1l(x) GENERIC x;{
  47         GENERIC z, d, s,c,ss,cc,r;
  85         int i, sgn;

  87         if (!finitel(x))
  88                 return (one / x);

  50         if (!finitel(x)) return one/x;
  90         sgn = signbitl(x);
  91         x = fabsl(x);

  93 #endif /* ! codereview */
  94         if (x > 1.28L) {
  95                 s = sinl(x);
  96                 c = cosl(x);

  98                 /* BEGIN CSTYLED */
  99 #endif /* ! codereview */
 100                 /*
 101                  * j1(x) = sqrt(2/(pi*x))*(p1(x)*cos(x0)-q1(x)*sin(x0))
 102                  * where x0 = x-3pi/4
 103                  *      Better formula:
 104                  *              cos(x0) = cos(x)cos(3pi/4)+sin(x)sin(3pi/4)
 105                  *                      =  1/sqrt(2) * (sin(x) - cos(x))
 106                  *              sin(x0) = sin(x)cos(3pi/4)-cos(x)sin(3pi/4)
 107                  *                      = -1/sqrt(2) * (cos(x) + sin(x))
 108                  * To avoid cancellation, use
 109                  *              sin(x) +- cos(x) = -cos(2x)/(sin(x) -+ cos(x))
 110                  * to compute the worse one.
 111                  */
 112                 /* END CSTYLED */
 113                 if (x > 1.0e2450L) {    /* x+x may overflow */
 114                         ss = -s - c;
 115                         cc = s - c;
  53                 if (x>1.0e2450L) {      /* x+x may overflow */
```

```
 54                         ss = -s-c;
 55                         cc =  s-c;
116                 } else if (signbitl(s) != signbitl(c)) {
117                         cc = s - c;
118                         ss = cosl(x + x) / cc;
 58                         ss = cosl(x+x)/cc;
119                 } else {
120                         ss = -s - c;
121                         cc = cosl(x + x) / ss;
 60                         ss = -s-c;
 61                         cc = cosl(x+x)/ss;
122                 }

124 #endif /* ! codereview */
125                 /*
126                  * j1(x) = 1/sqrt(pi*x) * (P(1,x)*cc - Q(1,x)*ss)
127                  * y1(x) = 1/sqrt(pi*x) * (P(1,x)*ss + Q(1,x)*cc)
128                  */
129                 if (x > 1.0e120L)
130                         return ((invsqrtpi * cc) / sqrtl(x));

132                 d = invsqrtpi * (pone(x) * cc - qone(x) * ss) / sqrtl(x);

134                 if (sgn == 0)
135                         return (d);
136                 else
137                         return (-d);
138         }

140         if (x <= small) {
141                 if (x <= tiny)
142                         d = 0.5L * x;
143                 else
144                         d = x * (0.5L - x * x * 0.125L);

146                 if (sgn == 0)
147                         return (d);
148                 else
149                         return (-d);
 63                 if (x>1.0e120L) return (invsqrtpi*cc)/sqrtl(x);
 64                 d =  invsqrtpi*(pone(x)*cc-qone(x)*ss)/sqrtl(x);
 65                 if (sgn == 0) return d; else return -d;
 66         }
 67         if (x<=small) {
 68             if (x<=tiny) d = 0.5L*x;
 69             else d =  x*(0.5L-x*x*0.125L);
 70             if (sgn == 0) return d; else return -d;
150         }

152         z = x * x;
 72         z = x*x;
153         r = r0[6];
154         s = s0[6];

156         for (i = 5; i >= 0; i--) {
157                 r = r * z + r0[i];
158                 s = s * z + s0[i];
 75             for (i=5;i>=0;i--) {
 76                 r = r*z + r0[i];
 77                 s = s*z + s0[i];
159         }

161         d = x * 0.5L + x * (z * (r / s));

163         if (sgn == 0)
164                 return (d);
```



```
165         else
166                 return (-d);
 79         d = x*0.5L+x*(z*(r/s));
 80         if (sgn == 0) return d; else return -d;
167 }
______unchanged_portion_omitted_

179 #endif /* ! codereview */
180 static const GENERIC v0[8] = {
181         1.0e0L,
182         1.369394302535807332517110204820556695644e-0002L,
183         9.508438148097659501433367062605935379588e-0005L,
184         4.399007309420092056052714797296467565655e-0007L,
185         1.488083087443756398305819693177715000787e-0009L,
186         3.751609832625793536245746965768587624922e-0012L,
187         6.680926434086257291872903276124244131448e-0015L,
188         6.676602383908906988160099057991121446058e-0018L,
189 };

191 GENERIC
192 y1l(GENERIC x)
193 {
194         GENERIC z, s, c, ss, cc, u, v;
 92 y1l(x) GENERIC x;{
 93         GENERIC z, s,c,ss,cc,u,v;
195         int i;

197         if (isnanl(x))
198                 return (x + x);

 96         if (isnanl(x)) return x+x;
200         if (x <= zero) {
201                 if (x == zero)
202                         return (-one / zero);
 99                     return -one/zero;
203                 else
204                         return (zero / zero);
101                     return zero/zero;
205         }

207 #endif /* ! codereview */
208         if (x > 1.28L) {
209                 if (!finitel(x))
210                         return (zero);

103                 if (!finitel(x)) return zero;
212                 s = sinl(x);
213                 c = cosl(x);

215                 /* BEGIN CSTYLED */
216 #endif /* ! codereview */
217                 /*
218                  * j1(x) = sqrt(2/(pi*x))*(p1(x)*cos(x0)-q1(x)*sin(x0))
219                  * where x0 = x-3pi/4
220                  *      Better formula:
221                  *              cos(x0) = cos(x)cos(3pi/4)+sin(x)sin(3pi/4)
222                  *                      =  1/sqrt(2) * (sin(x) - cos(x))
223                  *              sin(x0) = sin(x)cos(3pi/4)-cos(x)sin(3pi/4)
224                  *                      = -1/sqrt(2) * (cos(x) + sin(x))
225                  * To avoid cancellation, use
226                  *              sin(x) +- cos(x) = -cos(2x)/(sin(x) -+ cos(x))
227                  * to compute the worse one.
228                  */
229                 /* END CSTYLED */
230                 if (x > 1.0e2450L) {    /* x+x may overflow */
231                         ss = -s - c;
```

```
                        cc = s - c;
                if (x>1.0e2450L) {      /* x+x may overflow */
                        ss = -s-c;
                        cc =  s-c;
                } else if (signbitl(s) != signbitl(c)) {
                        cc = s - c;
                        ss = cosl(x + x) / cc;
                        ss = cosl(x+x)/cc;
                } else {
                        ss = -s - c;
                        cc = cosl(x + x) / ss;
                        ss = -s-c;
                        cc = cosl(x+x)/ss;
                }

#endif /* ! codereview */
                /*
                 * j1(x) = 1/sqrt(pi*x) * (P(1,x)*cc - Q(1,x)*ss)
                 * y1(x) = 1/sqrt(pi*x) * (P(1,x)*ss + Q(1,x)*cc)
                 */
                if (x > 1.0e91L)
                        return ((invsqrtpi * ss) / sqrtl(x));

                return (invsqrtpi * (pone(x) * ss + qone(x) * cc) / sqrtl(x));
                if (x>1.0e91L) return (invsqrtpi*ss)/sqrtl(x);
                return invsqrtpi*(pone(x)*ss+qone(x)*cc)/sqrtl(x);
        }

        if (x <= tiny)
                return (-tpi / x);

        z = x * x;
        u = u0[6];
        v = v0[6] + z * v0[7];

        for (i = 5; i >= 0; i--) {
                u = u * z + u0[i];
                v = v * z + v0[i];
        if (x<=tiny) {
            return (-tpi/x);
        }

        return (x * (u / v) + tpi * (j1l(x) * logl(x) - one / x));
        z = x*x;
            u = u0[6]; v = v0[6]+z*v0[7];
            for (i=5;i>=0;i--) {
                u = u*z + u0[i];
                v = v*z + v0[i];
            }
        return(x*(u/v) + tpi*(j1l(x)*logl(x)-one/x));
}
```

*unchanged_portion_omitted_*

```
#endif /* ! codereview */
static const GENERIC ps0[11] = {
        1.0e0L,
        1.06060068754589145560218084327675809510 7e+0003L,
        4.34310609341697558914715390650533890096 1e+0005L,
        8.91060586900217656658207224224435339905 9e+0007L,
        9.95912205863508788869071391762205654019 0e+0009L,
        6.18874496723494823179248294917104184389 4e+0011L,
        2.09886397695378350640175987380199030490 7e+0013L,
        3.67287035701806319674672975147993890845 0e+0014L,
        2.97553841924682492104901152957438588842 0e+0015L,
        9.06365765995043205018686029284479837091e+0015L,
        6.40195334431474791672936644150889271169 1e+0015L,
```



```
};

#endif /* ! codereview */
static const GENERIC pr1[12] = {
        1.00000000000000000000023667524130660984e+0000L,
        6.74615441997961875435480348812645297120 4e+0002L,
        1.81121078108339015485701833029614597050 2e+0005L,
        2.53309839037992426803800532909528784224 4e+0007L,
        2.02968361980534214525233857087542460072 9e+0009L,
        9.66085966219271146530106940159892998031 9e+0010L,
        2.74339623864483151993409896771662131631 6e+0012L,
        4.55309735414085437793102317026345524628 8e+0013L,
        4.21024506985221975747616986497487072037 4e+0014L,
        1.98733405622959648507664596717616980172 7e+0015L,
        4.06712005278709689383897045575133893046 2e+0015L,
        2.48653960638040639831084526491069139813 3e+0015L,
};

#endif /* ! codereview */
static const GENERIC ps1[14] = {
        1.0e0L,
        6.74498254497961875435580868019685952178 2e+0002L,
        1.81042179539696676203215529044136474035 0e+0005L,
        2.53098646064431065152958375969998843557 3e+0007L,
        2.02674327604802312136024928881829022414 5e+0009L,
        9.63746192440740593524526940705264134183 6e+0010L,
        2.73237862842376641740229279702831416083 1e+0012L,
        4.52234527496052712435484436401218427848 8e+0013L,
        4.16065066834174313268533575841546985654 5e+0014L,
        1.94373024298885820824349242489243590121 1e+0015L,
        3.88022853269212798990113161859806745000 1e+0015L,
        2.17802081616115461584100017368330299972 8e+0015L,
        -8.99406266684222555155434669817160063417 3e+0013L,
        1.36852036850885125349576480693461957499 0e+0013L,
};

#endif /* ! codereview */
static const GENERIC pr2[12] = {
        1.00000000000000000693865162184039623728 2e+0000L,
        3.65841629185040498140710107703794814469 8e+0002L,
        5.26707377217035654770979467060281244753 7e+0004L,
        3.91201210122683746301492521073589462044 2e+0006L,
        1.65129564897410395719387492871418076562 5e+0008L,
        4.11490114448079760997248499814214678349 9e+0009L,
        6.09252430976603668154298057252633514767 2e+0010L,
        5.26391317807128261671924996907413457057 7e+0011L,
        2.53840858112432422336734102053808133099 4e+0012L,
        6.28860792936029102789512698301536567764 8e+0012L,
        6.84833004821114841904705507538652594528 0e+0012L,
        2.29030964683886794142317816399142324469 0e+0012L,
};

#endif /* ! codereview */
static const GENERIC ps2[14] = {
        1.0e0L,
        3.65724441685040508645941016576231986185 6e+0002L,
        5.26280235842502324399238707586123730631 2e+0004L,
        3.90589681395991964813629586166148384836 4e+0006L,
        1.64679190779146122074269484210820277276 3e+0008L,
        4.09613280306425602222495412020820143734 4e+0009L,
        6.04666519591595044754442944573068023675 9e+0010L,
        5.19806173978199131341405221232865329516 8e+0011L,
        2.48423385181433396640152762642125427979 6e+0012L,
        6.04786880692531587933965153943431525594 0e+0012L,
        6.33310383125409165250164256729410181335 4e+0012L,
        1.87514309875428499446760993692468502496 8e+0012L,
```

```
        -5.2383309205633926929654127625088136015 34e+0010L,
        4.6568886094393647254277891983837792599 57e+0009L,
};

#endif /* ! codereview */
static const GENERIC pr3[13] = {
        1.0000000000000093368873180680561378428 97e+0000L,
        2.2427199427284595884880515720028357291 83e+0002L,
        1.9554506113820265502662577373310956910 92e+0004L,
        8.7071432939936198993954005624091755907 39e+0005L,
        2.1862678944870045659483242890109545053 16e+0007L,
        3.2243285105419577923606915856675028646 88e+0008L,
        2.8210573551513805973317928968827413648 97e+0009L,
        1.4453713872954224043655847937960289798 40e+0010L,
        4.1817431606698913577830110026565810786 4e+0010L,
        6.3873710887679931193255361377945355139 22e+0010L,
        4.5756199994127160780640705877674164363 96e+0010L,
        1.2284156512116391606202844416905035508 42e+0010L,
        7.2421703498755630534360505321531128820 72e+0008L,
};

#endif /* ! codereview */
static const GENERIC ps3[13] = {
        1.0e0L,
        2.2415480677285295510498046104860614010 70e+0002L,
        1.9528382167955521451321379329312371813 07e+0004L,
        8.6845749264931857446281273410699745755 26e+0005L,
        2.1763577710670379629408534128198521891 64e+0007L,
        3.1999586823561329773192587831671221005 67e+0008L,
        2.7862189315253346878446752199142018725 70e+0009L,
        1.4162837769517415496314175723179160397 67e+0010L,
        4.0429626592715679487356768346093488429 22e+0010L,
        6.0281684626466945100838472229684444021 61e+0010L,
        4.1184102267946414138338876065800852811 11e+0010L,
        9.9187357362970384307441612533382022302 63e+0009L,
        4.0929671982380980232191244874371303320 38e+0008L,
};

#endif /* ! codereview */
static const GENERIC pr4[13] = {
        1.0000000000015092209781573990420595533 90e+0000L,
        1.4375518683781478511334999632378260778 7e+0002L,
        7.9113355374181772960415180614045054280 04e+0003L,
        2.1937109391153172147165189089357561048 04e+0005L,
        3.3906624951367309625134897965382749843 82e+0006L,
        3.0486553479293488910060706092938842747 89e+0007L,
        1.6137816334894966063540451615274509751 95e+0008L,
        4.9750898350372302771101561500384821599 88e+0008L,
        8.6360470870151154038809044183395663232 64e+0008L,
        7.9182029123283661401106712230769491015 09e+0008L,
        3.4232946657989847334396503117227948532 94e+0008L,
        5.6219049534419639610405039347826626136 21e+0007L,
        2.0863035433102402607586704045094844997 93e+0006L,
};

#endif /* ! codereview */
static const GENERIC ps4[13] = {
        1.0e0L,
        1.4363799933845323716704933195918473623 04e+0002L,
        7.8946471547854306780610538488474366594 99e+0003L,
        2.1846597533920975290089817415508785861 74e+0005L,
        3.3661090833054651768035137381470494993 61e+0006L,
        3.0119115459689968176976658665872263431 86e+0007L,
        1.5822629137796898513167601484594148953 01e+0008L,
        4.8192688094943791921793858953013820177 0e+0008L,
        8.2013557629904506797028371234325271548 30e+0008L,
```



```
        7.2682320939825109374174464212823414252 12e+0008L,
        2.9509119090155729332621313239340364804 62e+0008L,
        4.2428399243059344230108589665406212193 96e+0007L,
        1.0643876204450907791821176663304051868 66e+0006L,
};

#endif /* ! codereview */
static const GENERIC pr5[13] = {
        1.0000000001024348052411714272538473538 61e+0000L,
        9.1293322570836292590605022490259632348 21e+0001L,
        3.1322384835869530375761193775045571914 13e+0003L,
        5.3297825282693079712789431224541711078 61e+0004L,
        4.9884601571841177906928730021030529441 45e+0005L,
        2.6866020716157868161470103342560474693 78e+0006L,
        8.4454185260289611977037998087012683018 31e+0006L,
        1.5365753586461411574757258899079008273 90e+0007L,
        1.5684058182365238217968627705865448119 45e+0007L,
        8.4508762398887701023876186673623021735 47e+0006L,
        2.1544149001395673284240268271632034460 77e+0006L,
        2.1056569265650438988884602548080623522 05e+0005L,
        4.7391650110233965070221343037368628129 75e+0003L,
};

#endif /* ! codereview */
static const GENERIC ps5[13] = {
        1.0e0L,
        9.1176135095953274765091526733947038477 93e+0001L,
        3.1216979724840156393012792292817707951 47e+0003L,
        5.2944472227358935680409118738345764402 55e+0004L,
        4.9303688821927723357982566841108878828 07e+0005L,
        2.6348546856411652983021674357983574377 68e+0006L,
        8.1854627754003263935558961570318182809 18e+0006L,
        1.4624174230802151926096686426630306670 86e+0007L,
        1.4506249939858516759828608441539548960 15e+0007L,
        7.4604676475619952832190865671620061138 64e+0006L,
        1.7542109814056124788692271425790563389 65e+0006L,
        1.4632867211552719715262649145247466995 96e+0005L,
        2.1558947257967020153412111165798270394 59e+0003L,
};

#endif /* ! codereview */
static const GENERIC pr6[13] = {
        1.0000000035648555467417359203157431571 29e+0000L,
        5.7340039348625404581194235099095102883 66e+0001L,
        1.2095724919358504860865596922917968879 76e+0003L,
        1.2433983914222812479336747791636602868 38e+0004L,
        6.9309967551814379372582209986017082787 87e+0004L,
        2.1980676595327575986467222499667676200 99e+0005L,
        4.0336594327120586339331791158205768584 55e+0005L,
        4.2577596572190080270160472065745743586 78e+0005L,
        2.5119173958760043494807212774457639163 89e+0005L,
        7.8137561530706236541787316513818819535 52e+0004L,
        1.1520691733811278813855880929058643528 91e+0004L,
        6.5485807828040885537778160375515233980 82e+0002L,
        8.6687253701169061323275427661279384968 80e+0000L,
};

#endif /* ! codereview */
static const GENERIC ps6[13] = {
        1.0e0L,
        5.7222852363571145664992215257362862051 84e+0001L,
        1.2030108428783179354445829506203395705 06e+0003L,
        1.2300583353785835501558255021724353712 08e+0004L,
        6.8009985506078612888653004386480898441 2e+0004L,
        2.1307678295993042629877693475368508859 21e+0005L,
        3.8404834666439166817599369729921553100 26e+0005L,
```

```
        3.947432373459225542861819148108081160393e+0005L,
        2.237816239393081111481588434457838526738e+0005L,
        6.545820495124419723398946273790921540774e+0004L,
        8.729563630320892741500726213278834737196e+0003L,
        4.130762660291894753450174794196998813709e+0002L,
        3.480368898672684645130335786015075595598e+0000L,
};

#endif /* ! codereview */
static const GENERIC sixteen = 16.0L;
static const GENERIC eight = 8.0L;
static const GENERIC huge = 1.0e30L;

static GENERIC
pone(GENERIC x)
static GENERIC pone(x)
GENERIC x;
{
        GENERIC s, r, t, z;
        GENERIC s,r,t,z;
        int i;

        if (x > huge)
                return (one);

        t = one / x;
        z = t * t;

        if (x > sixteen) {
                r = z * pr0[11] + pr0[10];
                s = ps0[10];

                for (i = 9; i >= 0; i--) {
                        r = z * r + pr0[i];
                        s = z * s + ps0[i];
                }
        } else if (x > eight) {
                r = pr1[11];
                s = ps1[11] + z * (ps1[12] + z * ps1[13]);

                for (i = 10; i >= 0; i--) {
                        r = z * r + pr1[i];
                        s = z * s + ps1[i];
                }
        } else if (x > five) {
                r = pr2[11];
                s = ps2[11] + z * (ps2[12] + z * ps2[13]);

                for (i = 10; i >= 0; i--) {
                        r = z * r + pr2[i];
                        s = z * s + ps2[i];
                }
        } else if (x > 3.5L) {
                r = pr3[12];
                s = ps3[12];

                for (i = 11; i >= 0; i--) {
                        r = z * r + pr3[i];
                        s = z * s + ps3[i];
                }
        } else if (x > 2.5L) {
                r = pr4[12];
                s = ps4[12];

                for (i = 11; i >= 0; i--) {
                        r = z * r + pr4[i];
```



```
                        s = z * s + ps4[i];
                }
        } else if (x > (1.0L / 0.5625L)) {
                r = pr5[12];
                s = ps5[12];

                for (i = 11; i >= 0; i--) {
                        r = z * r + pr5[i];
                        s = z * s + ps5[i];
        if (x>huge) return one;
        t = one/x; z = t*t;
        if (x>sixteen) {
            r = z*pr0[11]+pr0[10]; s = ps0[10];
            for (i=9;i>=0;i--) {
                r = z*r + pr0[i];
                s = z*s + ps0[i];
            }
        } else if (x>eight) {
            r = pr1[11]; s = ps1[11]+z*(ps1[12]+z*ps1[13]);
            for (i=10;i>=0;i--) {
                r = z*r + pr1[i];
                s = z*s + ps1[i];
            }
        } else if (x>five) {
            r = pr2[11]; s = ps2[11]+z*(ps2[12]+z*ps2[13]);
            for (i=10;i>=0;i--) {
                r = z*r + pr2[i];
                s = z*s + ps2[i];
            }
        } else if (x>3.5L) {
            r = pr3[12]; s = ps3[12];
            for (i=11;i>=0;i--) {
                r = z*r + pr3[i];
                s = z*s + ps3[i];
            }
        } else if (x>2.5L) {
            r = pr4[12]; s = ps4[12];
            for (i=11;i>=0;i--) {
                r = z*r + pr4[i];
                s = z*s + ps4[i];
            }
        } else if (x> (1.0L/0.5625L)) {
            r = pr5[12]; s = ps5[12];
            for (i=11;i>=0;i--) {
                r = z*r + pr5[i];
                s = z*s + ps5[i];
                }
        } else {                        /* assume x > 1.28 */
                r = pr6[12];
                s = ps6[12];

                for (i = 11; i >= 0; i--) {
                        r = z * r + pr6[i];
                        s = z * s + ps6[i];
            r = pr6[12]; s = ps6[12];
            for (i=11;i>=0;i--) {
                r = z*r + pr6[i];
                s = z*s + ps6[i];
                }
        }

        return (r / s);
        return r/s;
}
```

```
static const GENERIC qr0[12] = {
        3.749999999999999999999999999999999971033e-0001L,
        4.256726035237050601607682277433094262226e+0002L,
        1.875976490812878489192409978945401066066e+0005L,
        4.170314268048041914273603680317745592790e+0007L,
        5.092750132543855817293451118974555746551e+0009L,
        3.494749676278488654103505795794139483404e+0011L,
        1.327062148257437316997667817096694173709e+0013L,
        2.648993138273427226907503742066551150490e+0014L,
        2.511695665909547412222430494473998127684e+0015L,
        9.274694506662289043224310499164702306096e+0015L,
        8.150904170663663829331320302911792892002e+0015L,
        -5.001918733707662355772037829620388765122e+0014L,
};

#endif /* ! codereview */
static const GENERIC qs0[11] = {
        1.0e0L,
        1.135400380229880160428715273982155760093e+0003L,
        5.005701183877126164326765545516590744360e+0005L,
        1.113444200113712167984337603933040102987e+0008L,
        1.361074819925223062778717565699039471124e+0010L,
        9.355750985802849484438933905325982809653e+0011L,
        3.563462786008988825003965543857998084828e+0013L,
        7.155145113900094163648726863803802910454e+0014L,
        6.871266835834472758055559013851843654113e+0015L,
        2.622030899226736712644974988157345234092e+0016L,
        2.602912729172876330650077021706139707746e+0016L,
};

#endif /* ! codereview */
static const GENERIC qr1[12] = {
        3.749999999999999999997762458207284405806e-0001L,
        2.697883998881706839929255517498189980485e+0002L,
        7.755195925781028489386938870473834411019e+0004L,
        1.166777762104017777198211072895528968355e+0007L,
        1.011504772984321168320010084520261069362e+0009L,
        5.246007703574156853577754571720205550010e+0010L,
        1.637692549885592683166116551691266537647e+0012L,
        3.022303623698185669912990310925039382495e+0013L,
        3.154769927290655684846107030265909987946e+0014L,
        1.715819913441554770089730934808123360921e+0015L,
        4.165044355759732622273534445131736188510e+0015L,
        3.151381420874174705643100381708086287596e+0015L,
};

#endif /* ! codereview */
static const GENERIC qs1[14] = {
        1.0e0L,
        7.197091705351218239785633172408276982828e+0002L,
        2.070012799599548685544883041297609861055e+0005L,
        3.117014815317656221871840152778458754516e+0007L,
        2.705719678902554974863325877025902971727e+0009L,
        1.406113614727345726925060648750867264098e+0011L,
        4.403777536067131320363005978631674817359e+0012L,
        8.170725690209322283061499386703167242894e+0013L,
        8.609458844975495289227794126964431210566e+0014L,
        4.766766367015473481257280600694952920204e+0015L,
        1.202286587943342194863557940888115641650e+0016L,
        1.012474328306200909525063936061756024120e+0016L,
        6.183552022678917858273222879615824070703e+0014L,
        -9.756731548558226997573737400988225722740e+0013L,
};

#endif /* ! codereview */
static const GENERIC qr2[12] = {
```



```
        3.749999999999999481245647262226994293189e-0001L,
        1.471366807289771354491181140167359026735e+0002L,
        2.279432486768448220142080962843526951250e+0004L,
        1.828943048523771225163804043356958285893e+0006L,
        8.379828388647823135832220596417725010837e+0007L,
        2.279814029335044024585393671278378022053e+0009L,
        3.711653952257118120832817785271466441420e+0010L,
        3.557650914518554549916730572553105048068e+0011L,
        1.924583483146095896259774329498934160650e+0012L,
        5.424386256063736390759567088291887140278e+0012L,
        6.839325621241776786206509704671746841737e+0012L,
        2.702169563144001166291686452305436313971e+0012L,
};

#endif /* ! codereview */
static const GENERIC qs2[14] = {
        1.0e0L,
        3.926379194439388135703211933895203191089e+0002L,
        6.089148804106598297488336063007609312276e+0004L,
        4.893546162973278583711376356041614150645e+0006L,
        2.247571119114497845046388801813832219404e+0008L,
        6.137635663350177751290469334200757872645e+0009L,
        1.005115019784102856424493519524998953678e+0011L,
        9.725664462014503832860151384604677240620e+0011L,
        5.345525100485511116148634192844434636072e+0012L,
        1.549944007398946691720862738173956994779e+0013L,
        2.067148441178952625710302124163264760362e+0013L,
        9.401565402641963611295119487242595462301e+0012L,
        3.548217088622398274748837287769709374385e+0011L,
        -2.934470341719047120076509938432417352365e+0010L,
};

#endif /* ! codereview */
static const GENERIC qr3[13] = {
        3.749999999994127240845798332974514720 91e-0001L,
        9.058478580291706212422978492938435582527e+0001L,
        8.524056033161038750461083666711724381171e+0003L,
        4.105967158629109427753434569223631014730e+0005L,
        1.118326603378531348259783091972623333657e+0007L,
        1.794636683403578918528064904714132329343e+0008L,
        1.714314157463635959556133236004368896724e+0009L,
        9.622092032236084846572067257267661456030e+0009L,
        3.057759524485859159957762858780768355020e+0010L,
        5.129306780754798531609621454415938890020e+0010L,
        3.999122002794961070680636194346316041352e+0010L,
        1.122298454643493485989721564358100345388e+0010L,
        5.603981987645989709668830968522362582221e+0008L,
};

#endif /* ! codereview */
static const GENERIC qs3[13] = {
        1.0e0L,
        2.418328663076578169836155170053634419922e+0002L,
        2.279620205900121042587523541281272875520e+0004L,
        1.100984222585729521470129014992217092794e+0006L,
        3.010743223679247091004262516286654516282e+0007L,
        4.860925542827367817289619265215599433996e+0008L,
        4.686668111035348691982715864307839581243e+0009L,
        2.668701788405102017427214705946730894074e+0010L,
        8.677395746106802640390580944836650584903e+0010L,
        1.511936455574951790658498759541066430 36e+0011L,
        1.260845604432623478002018696873608353093e+0011L,
        4.052692278419853853911440231600864589805e+0010L,
        2.965516519212226064983267822243329694729e+0009L,
};
```

```
#endif /* ! codereview */
static const GENERIC qr4[13] = {
        3.749999999919234164154669754440123072618e-0001L,
        5.844218580776819864791168253485055101858e+0001L,
        3.489273514092912982675669411371435670220e+0003L,
        1.050523637774575684509663430018995479594e+0005L,
        1.764549172059701565500717319792780115289e+0006L,
        1.725532438844133795028063102681497371154e+0007L,
        9.938114847359778539965140247590176334874e+0007L,
        3.331710808184595545396883770200772842314e+0008L,
        6.271970557641881511609560444872797282698e+0008L,
        6.188529798677357075020774923903737913285e+0008L,
        2.821905302742849974509982167877885011629e+0008L,
        4.615467358646911976773290256984329814896e+0007L,
        1.348140608731546467396685802693380693275e+0006L,
};

#endif /* ! codereview */
static const GENERIC qs4[13] = {
        1.0e0L,
        1.561192663112345185261418296389902133372e+0002L,
        9.346678031144098270547225423124213083072e+0003L,
        2.825851246482293547838023847601704751590e+0005L,
        4.776572711622156091710902891124911556293e+0006L,
        4.715106953717135402977938048006267859302e+0007L,
        2.753962350894311316439652227611209035193e+0008L,
        9.428501434615463207768964787500411575223e+0008L,
        1.832650858775206787088236896454141572617e+0009L,
        1.901697378939743226948920874296595242257e+0009L,
        9.433322226854293780627188599226380812725e+0008L,
        1.808520540608671608680284520798858587370e+0008L,
        7.983342331736662753157217446919462398008e+0006L,
};

#endif /* ! codereview */
static const GENERIC qr5[13] = {
        3.749999995331364437028988850515190446719e-0001L,
        3.739356381766559882677514593041627547911e+0001L,
        1.399562500629413529355265462912819802551e+0003L,
        2.594154053098947925345332218062210111753e+0004L,
        2.640149879297408640394163979394594318371e+0005L,
        1.542471854873199142031889093591449397995e+0006L,
        5.242272868972053374067572098992335425895e+0006L,
        1.025834487769410221329633071426044839935e+0007L,
        1.116553924239448940142230579060124209622e+0007L,
        6.318076055959101763749163035258846535514e+0006L,
        1.641218086168640085276397359155128817 85e+0006L,
        1.522369793529178644168813882912134706444e+0005L,
        2.526530541062297200914180060208669584055e+0003L,
};

#endif /* ! codereview */
static const GENERIC qs5[13] = {
        1.0e0L,
        9.998960735935075380397545659016287506660e+0001L,
        3.758767417842043742686475060540416737562e+0003L,
        7.013652806952306520121959742519780781653e+0004L,
        7.208949808818615099246529616211730446850e+0005L,
        4.272753927109614455417836186072202009252e+0006L,
        1.482524411356470699336129814111025434703e+0007L,
        2.988750366665678233425279237627700803473e+0007L,
        3.396957890261080492694709150553619185065e+0007L,
        2.050652487738593004111578091156304540386e+0007L,
        5.900504120811732547616511555946279451316e+0006L,
        6.563391409260160897024498082273183468347e+0005L,
        1.692629845012790205348966731477187041419e+0004L,
```



```
};

#endif /* ! codereview */
static const GENERIC qr6[13] = {
        3.749999861516664133157566870858975421296e-0001L,
        2.367863756747764863120797431599473468918e+0001L,
        5.476715802114976248882067325630793143777e+0002L,
        6.143190357869842894025012945444096170251e+0003L,
        3.716250534677997850513733595140463851730e+0004L,
        1.270883463823876752138326905022875657430e+0005L,
        2.495301449636814481646371665429083801388e+0005L,
        2.789578988212952248340486296254398601942e+0005L,
        1.718247946911109055931819087137397324634e+0005L,
        5.458973214011665714330326732204106364229e+0004L,
        7.912102686687948786048943339759596652813e+0003L,
        4.077961006160866935722030715149087938091e+0002L,
        3.765206972770245085551057237882528510428e+0000L,
};

#endif /* ! codereview */
static const GENERIC qs6[13] = {
        1.0e0L, 6.341646532940517305641893852673926809601e+0001L,
    1.0e0L,
    6.341646532940517305641893852673926809601e+0001L,
        1.477058277414040790932597537920671025359e+0003L,
        1.674406564031044491436044253393536487604e+0004L,
        1.028516501369755949895050806908994650768e+0005L,
        3.593620042532885295087463507733285434207e+0005L,
        7.267924991381020915185873399453724799625e+0005L,
        8.462277510768818399961191426205006083088e+0005L,
        5.514399892230892163373611895645500250514e+0005L,
        1.898084241009259353540620272932188102299e+0005L,
        3.102941242117739015721984123081026253068e+0004L,
        1.958971184431466907681440650181421086143e+0003L,
        2.878853357310495087181721613889455121867e+0001L,
};

static GENERIC
qone(GENERIC x)
static GENERIC qone(x)
GENERIC x;
{
        GENERIC s, r, t, z;
        GENERIC s,r,t,z;
        int i;

        if (x > huge)
                return (0.375L / x);

        t = one / x;
        z = t * t;

        if (x > sixteen) {
                r = z * qr0[11] + qr0[10];
                s = qs0[10];

                for (i = 9; i >= 0; i--) {
                        r = z * r + qr0[i];
                        s = z * s + qs0[i];
                }
        } else if (x > eight) {
                r = qr1[11];
                s = qs1[11] + z * (qs1[12] + z * qs1[13]);

                for (i = 10; i >= 0; i--) {
                        r = z * r + qr1[i];
```

```
                        s = z * s + qs1[i];
                }
        } else if (x > five) {          /* x > 5.0 */
                r = qr2[11];
                s = qs2[11] + z * (qs2[12] + z * qs2[13]);

                for (i = 10; i >= 0; i--) {
                        r = z * r + qr2[i];
                        s = z * s + qs2[i];
                }
        } else if (x > 3.5L) {
                r = qr3[12];
                s = qs3[12];

                for (i = 11; i >= 0; i--) {
                        r = z * r + qr3[i];
                        s = z * s + qs3[i];
                }
        } else if (x > 2.5L) {
                r = qr4[12];
                s = qs4[12];

                for (i = 11; i >= 0; i--) {
                        r = z * r + qr4[i];
                        s = z * s + qs4[i];
                }
        } else if (x > (1.0L / 0.5625L)) {
                r = qr5[12];
                s = qs5[12];

                for (i = 11; i >= 0; i--) {
                        r = z * r + qr5[i];
                        s = z * s + qs5[i];
        if (x>huge) return 0.375L/x;
        t = one/x; z = t*t;
        if (x>sixteen) {
            r = z*qr0[11]+qr0[10]; s = qs0[10];
            for (i=9;i>=0;i--) {
                r = z*r + qr0[i];
                s = z*s + qs0[i];
            }
        } else if (x>eight) {
            r = qr1[11]; s = qs1[11]+z*(qs1[12]+z*qs1[13]);
            for (i=10;i>=0;i--) {
                r = z*r + qr1[i];
                s = z*s + qs1[i];
            }
        } else if (x>five) {    /* x > 5.0 */
            r = qr2[11]; s = qs2[11]+z*(qs2[12]+z*qs2[13]);
            for (i=10;i>=0;i--) {
                r = z*r + qr2[i];
                s = z*s + qs2[i];
            }
        } else if (x>3.5L) {
            r = qr3[12]; s = qs3[12];
            for (i=11;i>=0;i--) {
                r = z*r + qr3[i];
                s = z*s + qs3[i];
            }
        } else if (x>2.5L) {
            r = qr4[12]; s = qs4[12];
            for (i=11;i>=0;i--) {
                r = z*r + qr4[i];
                s = z*s + qs4[i];
            }
        } else if (x> (1.0L/0.5625L)) {
```



```
            r = qr5[12]; s = qs5[12];
            for (i=11;i>=0;i--) {
                r = z*r + qr5[i];
                s = z*s + qs5[i];
                }
        } else {                        /* assume x > 1.28 */
                r = qr6[12];
                s = qs6[12];

                for (i = 11; i >= 0; i--) {
                        r = z * r + qr6[i];
                        s = z * s + qs6[i];
            r = qr6[12]; s = qs6[12];
            for (i=11;i>=0;i--) {
                r = z*r + qr6[i];
                s = z*s + qs6[i];
                }
        }

        return (t * (r / s));
        return t*(r/s);
}
```

_____*unchanged_portion_omitted_*

```
**********************************************************
    7136 Thu Jun  6 20:38:48 2019
new/usr/src/lib/libm/common/Q/jnl.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __jnl = jnl
#pragma weak __ynl = ynl

/*
 * floating point Bessel's function of the 1st and 2nd kind
 * of order n: jn(n,x),yn(n,x);
 *
 * Special cases:
 *      y0(0)=y1(0)=yn(n,0) = -inf with division by zero signal;
 *      y0(-ve)=y1(-ve)=yn(n,-ve) are NaN with invalid signal.
 * Note 2. About jn(n,x), yn(n,x)
 *      For n=0, j0(x) is called,
 *      for n=1, j1(x) is called,
 *      for n<x, forward recursion us used starting
 *      from values of j0(x) and j1(x).
 *      for n>x, a continued fraction approximation to
 *      j(n,x)/j(n-1,x) is evaluated and then backward
 *      recursion is used starting from a supposed value
 *      for j(n,x). The resulting value of j(0,x) is
 *      compared with the actual value to correct the
 *      supposed value of j(n,x).
 *
 *      yn(n,x) is similar in all respects, except
 *      that forward recursion is used for all
 *      values of n>1.
 *
 */

#include "libm.h"
#include "longdouble.h"
```



```
#include <float.h>                      /* LDBL_MAX */

#define GENERIC long double

static const GENERIC
        invsqrtpi = 5.641895835477562869480794515607725858441e-0001L,
        two = 2.0L,
        zero = 0.0L,
        one = 1.0L;
invsqrtpi = 5.641895835477562869480794515607725858441e-0001L,
two  = 2.0L,
zero = 0.0L,
one  = 1.0L;

GENERIC
jnl(int n, GENERIC x)
{
        int i, sgn;
        GENERIC a, b, temp, z, w;

        /*
         * J(-n,x) = (-1)^n * J(n, x), J(n, -x) = (-1)^n * J(n, x)
         * Thus, J(-n,x) = J(n,-x)
         */
        if (n < 0) {
                n = -n;
                x = -x;
        }

#endif /* ! codereview */
        if (n == 0)
                return (j0l(x));

#endif /* ! codereview */
        if (n == 1)
                return (j1l(x));

#endif /* ! codereview */
        if (x != x)
                return (x + x);

        if ((n & 1) == 0)
                return (x+x);
        if ((n&1) == 0)
                sgn = 0;                /* even n */
        else
                sgn = signbitl(x);      /* old n  */

#endif /* ! codereview */
        x = fabsl(x);

        if (x == zero || !finitel(x)) {
                b = zero;
        } else if ((GENERIC)n <= x) {
        if (x == zero || !finitel(x)) b = zero;
        else if ((GENERIC)n <= x) {
                /*
                 * Safe to use
                 * J(n+1,x)=2n/x *J(n,x)-J(n-1,x)
                 */
                if (x > 1.0e91L) {
                        /*
                         * x >> n**2
                         *  Jn(x) = cos(x-(2n+1)*pi/4)*sqrt(2/x*pi)
                         *   Yn(x) = sin(x-(2n+1)*pi/4)*sqrt(2/x*pi)
                         *   Let s=sin(x), c=cos(x),
```

```
 120                         *      xn=x-(2n+1)*pi/4, sqt2 = sqrt(2),then
 121                         *
 122                         *         n    sin(xn)*sqt2    cos(xn)*sqt2
 123                         *      ----------------------------------
 124                         *         0     s-c             c+s
 125                         *         1    -s-c            -c+s
 126                         *         2    -s+c            -c-s
 127                         *         3     s+c             c-s
 128                         */
 129                         switch (n & 3) {
  70                         switch (n&3) {
 130                         case 0:
 131                                 temp = cosl(x) + sinl(x);
  72                                 temp =  cosl(x)+sinl(x);
 132                                 break;
 133                         case 1:
 134                                 temp = -cosl(x) + sinl(x);
  75                                 temp = -cosl(x)+sinl(x);
 135                                 break;
 136                         case 2:
 137                                 temp = -cosl(x) - sinl(x);
  78                                 temp = -cosl(x)-sinl(x);
 138                                 break;
 139                         case 3:
 140                                 temp = cosl(x) - sinl(x);
  81                                 temp =  cosl(x)-sinl(x);
 141                                 break;
 142                         }

 144                         b = invsqrtpi * temp / sqrtl(x);
  84                         b = invsqrtpi*temp/sqrtl(x);
 145                 } else {
 146                         a = j0l(x);
 147                         b = j1l(x);

 149 #endif /* ! codereview */
 150                         for (i = 1; i < n; i++) {
 151                                 temp = b;
 152                                 /* avoid underflow */
 153                                 b = b * ((GENERIC)(i + i) / x) - a;
  88                                 b = b*((GENERIC)(i+i)/x) - a;
 154                                 a = temp;
 155                         }
 156                 }
 157         } else {
 158                 if (x < 1e-17L) {       /* use J(n,x) = 1/n!*(x/2)^n */
 159                         b = powl(0.5L * x, (GENERIC)n);

  94                         b = powl(0.5L*x, (GENERIC)n);
 161                         if (b != zero) {
 162                                 for (a = one, i = 1; i <= n; i++)
 163                                         a *= (GENERIC)i;

 165                                 b = b / a;
  98                                 b = b/a;
 166                         }
 167                 } else {
 168                         /* use backward recurrence */
 169                         /* BEGIN CSTYLED */
 170                         /*
 171                          *                        x      x^2      x^2
 172                          *  J(n,x)/J(n-1,x) =  ----   ------   ------   .....
 173                          *                        2n  - 2(n+1) - 2(n+2)
 174                          *
 175                          *                        1      1        1
 176                          *  (for large x)   =  ----  ------   ------   .....
```



```
 177                          *                      2n   2(n+1)   2(n+2)
 178                          *                      -- - ------ - ------ -
 179                          *                       x     x         x
 180                          *
 181                          * Let w = 2n/x and h=2/x, then the above quotient
 182                          * is equal to the continued fraction:
 183                          *                 1
 184                          *      = -----------------------
 185                          *                     1
 186                          *         w - -----------------
 187                          *                       1
 188                          *              w+h - ---------
 189                          *                     w+2h - ...
 190                          *
 191                          * To determine how many terms needed, let
 192                          * Q(0) = w, Q(1) = w(w+h) - 1,
 193                          * Q(k) = (w+k*h)*Q(k-1) - Q(k-2),
 194                          * When Q(k) > 1e4      good for single
 195                          * When Q(k) > 1e9      good for double
 196                          * When Q(k) > 1e17     good for quaduple
 197                          */
 198                         /* END CSTYLED */
 199                         /* determine k */
 200                         GENERIC t, v;
 201                         double q0, q1, h, tmp;
 202                         int k, m;

 204                         w = (n + n) / (double)x;
 205                         h = 2.0 / (double)x;
 136                         w  = (n+n)/(double)x;
 137                         h = 2.0/(double)x;
 206                         q0 = w;
 207                         z = w + h;
 208                         q1 = w * z - 1.0;
 139                         z = w+h;
 140                         q1 = w*z - 1.0;
 209                         k = 1;

 211 #endif /* ! codereview */
 212                         while (q1 < 1.0e17) {
 213                                 k += 1;
 214                                 z += h;
 215                                 tmp = z * q1 - q0;
 142                                 tmp = z*q1 - q0;
 216                                 q0 = q1;
 217                                 q1 = tmp;
 218                         }

 220                         m = n + n;

 222                         for (t = zero, i = 2 * (n + k); i >= m; i -= 2)
 223                                 t = one / (i / x - t);

 146                         m = n+n;
 147                         for (t = zero, i = 2*(n+k); i >= m; i -= 2)
 148                                 t = one/(i/x-t);
 225                         a = t;
 226                         b = one;

 228 #endif /* ! codereview */
 229                         /*
 230                          * estimate log((2/x)^n*n!) = n*log(2/x)+n*ln(n)
 231                          * hence, if n*(log(2n/x)) > ...
 232                          *  single:
 233                          *    8.8722839355e+01
 234                          *  double:
```

```
 235                         *    7.09782712893383973096e+02
 236                         *  long double:
 237                         *    1.13565234062941439494919310779707650 06170e+04
 238                         *  then recurrent value may overflow and the result is
 239                         *  likely underflow to zero
 240                         */
 241                        tmp = n;
 242                        v = two / x;
 243                        tmp = tmp * logl(fabsl(v * tmp));

 151                        v = two/x;
 152                        tmp = tmp*logl(fabsl(v*tmp));
 245                        if (tmp < 1.135652340629414394949193107797076 5e+04L) {
 246                                for (i = n - 1; i > 0; i--) {
 154                                for (i = n-1; i > 0; i--) {
 247                                        temp = b;
 248                                        b = ((i + i) / x) * b - a;
 156                                        b = ((i+i)/x)*b - a;
 249                                        a = temp;
 250                                }
 251                        } else {
 252                                for (i = n - 1; i > 0; i--) {
 160                                for (i = n-1; i > 0; i--) {
 253                                        temp = b;
 254                                        b = ((i + i) / x) * b - a;
 162                                        b = ((i+i)/x)*b - a;
 255                                        a = temp;

 257 #endif /* ! codereview */
 258                                        if (b > 1e1000L) {
 259                                                a /= b;
 260                                                t /= b;
 261                                                b = 1.0;
 262                                        }
 263                                }
 264                        }

 266                        b = (t * j0l(x) / b);
 164                        b = (t*j0l(x)/b);
 267                }
 268        }

 270 #endif /* ! codereview */
 271        if (sgn != 0)
 272                return (-b);
 273        else
 274                return (b);
 275 }

 277 GENERIC
 278 ynl(int n, GENERIC x)
 279 {
 280        int i;
 281        int sign;
 282        GENERIC a, b, temp;

 284        if (x != x)
 285                return (x + x);

 167                return (x+x);
 287        if (x <= zero) {
 288                if (x == zero)
 289                        return (-one / zero);
 170                        return (-one/zero);
 290                else
 291                        return (zero / zero);
```



```
 172                        return (zero/zero);
 292        }

 294 #endif /* ! codereview */
 295        sign = 1;

 297 #endif /* ! codereview */
 298        if (n < 0) {
 299                n = -n;

 301                if ((n & 1) == 1)
 302                        sign = -1;
 174                if ((n&1) == 1) sign = -1;
 303        }

 305 #endif /* ! codereview */
 306        if (n == 0)
 307                return (y0l(x));

 309 #endif /* ! codereview */
 310        if (n == 1)
 311                return (sign * y1l(x));

 176                return (sign*y1l(x));
 313        if (!finitel(x))
 314                return (zero);

 316        if (x > 1.0e91L) {
 317                /*
 318                 * x >> n**2
 319                 *   Jn(x) = cos(x-(2n+1)*pi/4)*sqrt(2/x*pi)
 320                 *   Yn(x) = sin(x-(2n+1)*pi/4)*sqrt(2/x*pi)
 321                 *   Let s = sin(x), c = cos(x),
 322                 *      xn = x-(2n+1)*pi/4, sqt2 = sqrt(2), then
 323                 *
 324                 *         n    sin(xn)*sqt2    cos(xn)*sqt2
 325                 *      ----------------------------------
 326                 *         0     s-c             c+s
 327                 *         1    -s-c            -c+s
 328                 *         2    -s+c            -c-s
 329                 *         3     s+c             c-s
 330                 */
 331                switch (n & 3) {
 195                switch (n&3) {
 332                case 0:
 333                        temp = sinl(x) - cosl(x);
 197                        temp =  sinl(x)-cosl(x);
 334                        break;
 335                case 1:
 336                        temp = -sinl(x) - cosl(x);
 200                        temp = -sinl(x)-cosl(x);
 337                        break;
 338                case 2:
 339                        temp = -sinl(x) + cosl(x);
 203                        temp = -sinl(x)+cosl(x);
 340                        break;
 341                case 3:
 342                        temp = sinl(x) + cosl(x);
 206                        temp =  sinl(x)+cosl(x);
 343                        break;
 344                }

 346                b = invsqrtpi * temp / sqrtl(x);
 209                b = invsqrtpi*temp/sqrtl(x);
 347        } else {
 348                a = y0l(x);
```

```
                b = y1l(x);

#endif /* ! codereview */
                /*
                 * fix 1262058 and take care of non-default rounding
                 */
                for (i = 1; i < n; i++) {
                        temp = b;
                        b *= (GENERIC)(i + i) / x;

                        b *= (GENERIC) (i + i) / x;
                        if (b <= -LDBL_MAX)
                                break;

#endif /* ! codereview */
                        b -= a;
                        a = temp;
                }
        }

#endif /* ! codereview */
        if (sign > 0)
                return (b);
        else
                return (-b);
}
```

```
**********************************************************
    1325 Thu Jun  6 20:38:48 2019
new/usr/src/lib/libm/common/Q/lgammal.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __lgammal = lgammal

/*
 * long double lgammal(long double x);
 */

#include "libm.h"
#include "longdouble.h"

extern int signgam;
extern int signgaml;

long double
lgammal(long double x)
{
lgammal(long double x) {
        long double y = __k_lgammal(x, &signgaml);

        signgam = signgaml;     /* SUSv3 requires the setting of signgam */
        return (y);
}
______unchanged_portion_omitted_
```

```
**********************************************************
    1238 Thu Jun  6 20:38:49 2019
new/usr/src/lib/libm/common/Q/lgammal_r.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

/*
 * long double lgammal_r(long double x, int *signgamlp);
 */

#pragma weak __lgammal_r = lgammal_r

#include "libm.h"
#include "longdouble.h"

long double
lgammal_r(long double x, int *signgamlp)
{
lgammal_r(long double x, int *signgamlp) {
        return (__k_lgammal(x, signgamlp));
}
______unchanged_portion_omitted_
```

```
**********************************************************
    3055 Thu Jun  6 20:38:49 2019
new/usr/src/lib/libm/common/Q/log10l.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __log10l = log10l

/*
 * log10l(X)
 *
 * Method :
 *      Let log10_2hi = leading 98(SPARC)/49(x86) bits of log10(2) and
 *          log10_2lo = log10(2) - log10_2hi,
 *          ivln10   = 1/log(10) rounded.
 *      Then
 *              n = ilogb(x),
 *              if (n<0)  n = n+1;
 *              x = scalbn(x,-n);
 *              LOG10(x) := n*log10_2hi + (n*log10_2lo + ivln10*log(x))
 *
 * Note1:
 *      For fear of destroying log10(10**n)=n, the rounding mode is
 *      set to Round-to-Nearest.
 *
 * Special cases:
 *      log10(x) is NaN with signal if x < 0;
 *      log10(+INF) is +INF with no signal; log10(0) is -INF with signal;
 *      log10(NaN) is that NaN with no signal;
 *      log10(10**N) = N  for N=0,1,...,22.
 *
 * Constants:
 * The hexadecimal values are the intended ones for the following constants.
 * The decimal values may be used, provided that the compiler will convert
 * from decimal to binary accurately enough to produce the hexadecimal values
 * shown.
```



```
 */

#include "libm.h"
#include "longdouble.h"

#if defined(__x86)
#define __swapRD        __swap87RD
#endif

#endif /* ! codereview */
extern enum fp_direction_type __swapRD(enum fp_direction_type);
static const long double zero = 0.0L,

static const long double
        zero      = 0.0L,
        ivln10 = 4.342944819032518276511289189166050822944e-0001L,
        one = 1.0L,
#if defined(__x86)
        log10_2hi = 3.010299956639803653501985536422580480576e-01L,
        log10_2lo = 8.298635403410822349787106337291183585413e-16L;
#elif defined(__sparc)
        log10_2hi = 3.010299956639811952137388947242098603469e-01L,
        log10_2lo = 2.831664213089468167896664371953210945664e-31L;
#else
#error Unknown Architecture!
#endif

long double
log10l(long double x)
{
log10l(long double x) {
        long double y, z;
        enum fp_direction_type rd;
        int n;

        if (!finitel(x)) {
        if (!finitel(x))
                return (x + fabsl(x));  /* x is +-INF or NaN */
        } else if (x > zero) {
        else if (x > zero) {
                n = ilogbl(x);

#endif /* ! codereview */
                if (n < 0)
                        n += 1;

#endif /* ! codereview */
                rd = __swapRD(fp_nearest);
                y = n;
                x = scalbnl(x, -n);
                z = y * log10_2lo + ivln10 * logl(x);
                z += y * log10_2hi;

#endif /* ! codereview */
                if (rd != fp_nearest)
                        (void) __swapRD(rd);

#endif /* ! codereview */
                return (z);
        } else if (x == zero) {         /* -INF */
        } else if (x == zero)   /* -INF */
                return (-one / zero);
        } else {                        /* x <0, return NaN */
        else                    /* x <0, return NaN */
                return (zero / zero);
        }
```

```
#endif /* ! codereview */
}
```

```
**********************************************************
    6980 Thu Jun  6 20:38:50 2019
new/usr/src/lib/libm/common/Q/log1pl.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#ifdef __LITTLE_ENDIAN
#define H0(x)           *(3 + (int *)&x)
#define H1(x)           *(2 + (int *)&x)
#define H2(x)           *(1 + (int *)&x)
#define H3(x)           *(int *)&x
#define H0(x)   *(3 + (int *) &x)
#define H1(x)   *(2 + (int *) &x)
#define H2(x)   *(1 + (int *) &x)
#define H3(x)   *(int *) &x
#else
#define H0(x)           *(int *)&x
#define H1(x)           *(1 + (int *)&x)
#define H2(x)           *(2 + (int *)&x)
#define H3(x)           *(3 + (int *)&x)
#define H0(x)   *(int *) &x
#define H1(x)   *(1 + (int *) &x)
#define H2(x)   *(2 + (int *) &x)
#define H3(x)   *(3 + (int *) &x)
#endif

/*
 * log1pl(x)
 * Table look-up algorithm by modifying logl.c
 * By K.C. Ng, July 6, 1995
 *
 * (a). For 1+x in [31/33,33/31], using a special approximation:
 *      s = x/(2.0+x);  ... here |s| <= 0.03125
 *      z = s*s;
 *      return x-s*(x-z*(B1+z*(B2+z*(B3+z*(B4+...+z*B9)...))));
 *      (i.e., x is in [-2/33,2/31])
```



```
 *
 * (b). Otherwise, normalize 1+x = 2^n * 1.f.
 *      Here we may need a correction term for 1+x rounded.
 *      Use a 6-bit table look-up: find a 6 bit g that match f to 6.5 bits,
 *      then
 *          log(1+x) = n*ln2 + log(1.g) + log(1.f/1.g).
 *      Here the leading and trailing values of log(1.g) are obtained from
 *      a size-64 table.
 *      For log(1.f/1.g), let s = (1.f-1.g)/(1.f+1.g). Note that
 *              1.f = 2^-n(1+x)
 *
 *      then
 *          log(1.f/1.g) = log((1+s)/(1-s)) = 2s + 2/3 s^3 + 2/5 s^5 +...
 *      Note that |s|<2**-8=0.00390625. We use an odd s-polynomial
 *      approximation to compute log(1.f/1.g):
 *              s*(A1+s^2*(A2+s^2*(A3+s^2*(A4+s^2*(A5+s^2*(A6+s^2*A7))))))
 *      (Precision is 2**-136.91 bits, absolute error)
 *
 *      CAUTION:
 *      For x>=1, compute 1+x will lost one bit (OK).
 *      For x in [-0.5,-1), 1+x is exact.
 *      For x in (-0.5,-2/33]U[2/31,1), up to 4 last bits of x will be lost
 *      in 1+x.  Therefore, to recover the lost bits, one need to compute
 *      1.f-1.g accurately.
 *
 *      Let hx = HI(x), m = (hx>>16)-0x3fff (=ilogbl(x)), note that
 *              -2/33 = -0.0606...= 2^-5 * 1.939...,
 *               2/31 =  0.09375  = 2^-4 * 1.500...,
 *      so for x in (-0.5,-2/33], -5<=m<=-2,  n= -1, 1+f=2*(1+x)
 *         for x in [2/33,1),     -4<=m<=-1,  n=  0, f=x
 *
 *      In short:
 *      if x>0, let g: hg= ((hx + (0x200<<(-m)))>>(10-m))<<(10-m)
 *      then 1.f-1.g = x-g
 *      if x<0, let g': hg' =((ix-(0x200)<<(-m-1))>>(9-m))<<(9-m)
 *      (ix=hx&0x7fffffff)
 *      then 1.f-1.g = 2*(g'+x),
 *
 * (c). The final result is computed by
 *              (n*ln2_hi+_TBL_logl_hi[j]) +
 *                      ( (n*ln2_lo+_TBL_logl_lo[j]) + s*(A1+...) )
 *
 * Note.
 *      For ln2_hi and _TBL_logl_hi[j], we force their last 32 bit to be zero
 *      so that n*ln2_hi + _TBL_logl_hi[j] is exact. Here
 *      _TBL_logl_hi[j] + _TBL_logl_lo[j] match log(1+j*2**-6) to 194 bits
 *
 *
 * Special cases:
 *      log(x) is NaN with signal if x < 0 (including -INF) ;
 *      log(+INF) is +INF; log(0) is -INF with signal;
 *      log(NaN) is that NaN with no signal.
 *
 * Constants:
 * The hexadecimal values are the intended ones for the following constants.
 * The decimal values may be used, provided that the compiler will convert
 * from decimal to binary accurately enough to produce the hexadecimal values
 * shown.
 */

#pragma weak __log1pl = log1pl

#include "libm.h"

extern const long double _TBL_logl_hi[], _TBL_logl_lo[];
static const long double zero = 0.0L,
```

```
        one = 1.0L,
        two = 2.0L,
        ln2hi = 6.931471805599453094172319547495844850203e-0001L,
        ln2lo = 1.667085920830552208890449330400379754169e-0025L,
        A1 = 2.000000000000000000000000000000000000024e+0000L,
        A2 = 6.666666666666666666666666666666091393804e-0001L,
        A3 = 4.000000000000000000000000000407167070220671e-0001L,
        A4 = 2.857142857142857142730077490612903681164e-0001L,
        A5 = 2.222222222222242577702836920812882605099e-0001L,
        A6 = 1.818181816435493395985912667105885828356e-0001L,
        A7 = 1.538537835211839751112067512805496931725e-0001L,
        B1 = 6.666666666666666666666666666666961498329e-0001L,
        B2 = 3.999999999999999999999999999037655042358e-0001L,
        B3 = 2.857142857142857142857273426428347457918e-0001L,
        B4 = 2.222222222222222221353229049747910109566e-0001L,
        B5 = 1.818181818181821503532559306309070138046e-0001L,
        B6 = 1.538461538453809210486356084587356788556e-0001L,
        B7 = 1.333333344463358756121456892645178795480e-0001L,
        B8 = 1.176460904783899064854645174603360383792e-0001L,
        B9 = 1.057293869956598995326368602518056990746e-0001L;

static const long double
zero    =   0.0L,
one     =   1.0L,
two     =   2.0L,
ln2hi   =   6.931471805599453094172319547495844850203e-0001L,
ln2lo   =   1.667085920830552208890449330400379754169e-0025L,
A1      =   2.000000000000000000000000000000000000024e+0000L,
A2      =   6.666666666666666666666666666666091393804e-0001L,
A3      =   4.000000000000000000000000000407167070220671e-0001L,
A4      =   2.857142857142857142730077490612903681164e-0001L,
A5      =   2.222222222222242577702836920812882605099e-0001L,
A6      =   1.818181816435493395985912667105885828356e-0001L,
A7      =   1.538537835211839751112067512805496931725e-0001L,
B1      =   6.666666666666666666666666666666961498329e-0001L,
B2      =   3.999999999999999999999999999037655042358e-0001L,
B3      =   2.857142857142857142857273426428347457918e-0001L,
B4      =   2.222222222222222221353229049747910109566e-0001L,
B5      =   1.818181818181821503532559306309070138046e-0001L,
B6      =   1.538461538453809210486356084587356788556e-0001L,
B7      =   1.333333344463358756121456892645178795480e-0001L,
B8      =   1.176460904783899064854645174603360383792e-0001L,
B9      =   1.057293869956598995326368602518056990746e-0001L;

long double
log1pl(long double x)
{
log1pl(long double x) {
        long double f, s, z, qn, h, t, y, g;
        int i, j, ix, iy, n, hx, m;

        hx = H0(x);
        ix = hx & 0x7fffffff;

#endif /* ! codereview */
        if (ix < 0x3ffaf07c) {          /* |x|<2/33 */
                if (ix <= 0x3f8d0000) { /* x <= 2**-114, return x */
                        if ((int)x == 0)
                        if ((int) x == 0)
                                return (x);
                }

#endif /* ! codereview */
                s = x / (two + x);      /* |s|<2**-8 */
                z = s * s;
                return (x - s * (x - z * (B1 + z * (B2 + z * (B3 + z * (B4 + z *
```



```
                        (B5 + z * (B6 + z * (B7 + z * (B8 + z * B9)))))))))));
                return (x - s * (x - z * (B1 + z * (B2 + z * (B3 + z * (B4 +
                        z * (B5 + z * (B6 + z * (B7 + z * (B8 + z * B9)))))))))));
        }

        if (ix >= 0x7fff0000)           /* x is +inf or NaN */
        if (ix >= 0x7fff0000) { /* x is +inf or NaN */
                return (x + fabsl(x));

        }
        if (hx < 0 && ix >= 0x3fff0000) {
                if (ix > 0x3fff0000 || (H1(x) | H2(x) | H3(x)) != 0)
                        x = zero;

#endif /* ! codereview */
                return (x / zero);      /* log1p(x) is NaN  if x<-1 */
                /* log1p(-1) is -inf */
        }

#endif /* ! codereview */
        if (ix >= 0x7ffeffff)
                y = x;                  /* avoid spurious overflow */
        else
                y = one + x;

#endif /* ! codereview */
        iy = H0(y);
        n = ((iy + 0x200) >> 16) - 0x3fff;
        iy = (iy & 0x0000ffff) | 0x3fff0000;    /* scale 1+x to [1,2] */
        H0(y) = iy;
        z = zero;
        m = (ix >> 16) - 0x3fff;

#endif /* ! codereview */
        /* HI(1+x) = (((hx&0xffff)|0x10000)>>(-m))|0x3fff0000 */
        if (n == 0) {                   /* x in [2/33,1) */
                g = zero;
                H0(g) = ((hx + (0x200 << (-m))) >> (10 - m)) << (10 - m);
                t = x - g;
                i = (((((hx & 0xffff) | 0x10000) >> (-m)) | 0x3fff0000) +
                        0x200) >> 10;
                H0(z) = i << 10;

        } else if ((1 + n) == 0 && (ix < 0x3ffe0000)) { /* x in (-0.5,-2/33] */
                g = zero;
                H0(g) = ((ix + (0x200 << (-m - 1))) >> (9 - m)) << (9 - m);
                t = g + x;
                t = t + t;

#endif /* ! codereview */
                /*
                 * HI(2*(1+x)) =
                 * ((0x10000-(((hx&0xffff)|0x10000)>>(-m)))<<1)|0x3fff0000
                 */

#endif /* ! codereview */
                /*
                 * i =
                 * ((((0x10000-(((hx&0xffff)|0x10000)>>(-m)))<<1)|0x3fff0000)+
                 * 0x200)>>10; H0(z)=i<<10;
                 */
                z = two * (one - g);
                i = H0(z) >> 10;
        } else {
                i = (iy + 0x200) >> 10;
                H0(z) = i << 10;
```

```
                t = y - z;
        }

        s = t / (y + z);
        j = i & 0x3f;
        z = s * s;
        qn = (long double)n;
        qn = (long double) n;
        t = qn * ln2lo + _TBL_logl_lo[j];
        h = qn * ln2hi + _TBL_logl_hi[j];
        f = t + s * (A1 + z * (A2 + z * (A3 + z * (A4 + z * (A5 + z * (A6 + z *
            A7))))));
        f = t + s * (A1 + z * (A2 + z * (A3 + z * (A4 + z * (A5 + z * (A6 +
                z * A7))))));
        return (h + f);
}
```

_____*unchanged_portion_omitted_*

```
**********************************************************
    1610 Thu Jun  6 20:38:50 2019
new/usr/src/lib/libm/common/Q/log2l.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

/*
 * log2l(x)
 * RETURN THE BASE 2 LOGARITHM OF X
 *
 * Method:
 *      purge off 0,INF, and NaN.
 *      n = ilogb(x)
 *      if (n<0) n+=1
 *      z = scalbn(x,-n)
 *      LOG2(x) = n + (1/ln2)*log(x)
 */

#pragma weak __log2l = log2l

#include "libm.h"
#include "longdouble.h"

static const long double zero = 0.0L,
static const long double
        zero    = 0.0L,
        half = 0.5L,
        one = 1.0L,
        invln2 = 1.4426950408889634073599246810018921374 27e+0000L;

long double
log2l(long double x)
{
log2l(long double x) {
        int n;
```



```
        if (x == zero || !finitel(x))
                return (logl(x));

#endif /* ! codereview */
        n = ilogbl(x);

#endif /* ! codereview */
        if (n < 0)
                n += 1;

#endif /* ! codereview */
        x = scalbnl(x, -n);

#endif /* ! codereview */
        if (x == half)
                return (n - one);

#endif /* ! codereview */
        return (n + invln2 * logl(x));
}
```

```
**********************************************************
    2470 Thu Jun  6 20:38:51 2019
new/usr/src/lib/libm/common/Q/logbl.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __logbl = logbl

#include "libm.h"
#include "xpg6.h"                       /* __xpg6 */
#define _C99SUSv3_logb          _C99SUSv3_logb_subnormal_is_like_ilogb

#if defined(__sparc)
#define ISNORMALL(k, x)         (k != 0x7fff)   /* assuming k != 0 */
#define X86PDNRM(k, x)
#define XSCALE_OFFSET           0x406f          /* 0x3fff + 112 */

#endif /* ! codereview */
static const long double xscale = 5192296858534827628530496329220096.0L;

/* 2^112 */
                                                                        /* 2^112 */
#elif defined(__x86)
/*
 * if pseudo-denormal, replace by the equivalent normal
 */
#define X86PDNRM(k, x)  \
        if (k == 0 && (((int *)&x)[1] & 0x80000000) !=  0) \
                ((int *)&x)[2] |= k = 1
#define X86PDNRM(k, x)  if (k == 0 && (((int *) &x)[1] & 0x80000000) != 0) \
                                ((int *) &x)[2] |= k = 1
#if defined(HANDLE_UNSUPPORTED)         /* assuming k != 0 */
#define ISNORMALL(k, x) (k != 0x7fff && (((int *)&x)[1] & 0x80000000) != \
        0)
#define ISNORMALL(k, x) (k != 0x7fff && (((int *) &x)[1] & 0x80000000) != 0)
#else
```



```
#define ISNORMALL(k, x) (k != 0x7fff)
#endif
#define XSCALE_OFFSET   0x403e                          /* 0x3fff + 63 */
static const long double xscale = 9223372036854775808.0L;       /* 2^63 */
#endif

static long double
raise_division(long double v)
{
raise_division(long double v) {
#pragma STDC FENV_ACCESS ON

#endif /* ! codereview */
        static const long double zero = 0.0L;

#endif /* ! codereview */
        return (v / zero);
}

long double
logbl(long double x)
{
logbl(long double x) {
        int k = XBIASED_EXP(x);

        X86PDNRM(k, x);

#endif /* ! codereview */
        if (k == 0) {
                if (ISZEROL(x)) {
                if (ISZEROL(x))
                        return (raise_division(-1.0L));
                } else if ((__xpg6 & _C99SUSv3_logb) != 0) {
                else if ((__xpg6 & _C99SUSv3_logb) != 0) {
                        x *= xscale;    /* scale up by 2^112 or 2^63 */
                        return ((long double)(XBIASED_EXP(x) - XSCALE_OFFSET));
                } else {
                        return (long double) (XBIASED_EXP(x) - XSCALE_OFFSET);
                } else
                        return (-16382.L);
                }
        } else if (ISNORMALL(k, x)) {
                return ((long double)(k - 0x3fff));
        } else {
        } else if (ISNORMALL(k, x))
                return ((long double) (k - 0x3fff));
        else
                return (x * x);
        }
#endif /* ! codereview */
}
```

```
**********************************************************
    5478 Thu Jun  6 20:38:51 2019
new/usr/src/lib/libm/common/Q/logl.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __logl = logl

/*
 * logl(x)
 * Table look-up algorithm
 * By K.C. Ng, March 6, 1989
 *
 * (a). For x in [31/33,33/31], using a special approximation:
 *      f = x - 1;
 *      s = f/(2.0+f);  ... here |s| <= 0.03125
 *      z = s*s;
 *      return f-s*(f-z*(B1+z*(B2+z*(B3+z*(B4+...+z*B9)...))));
 *
 * (b). Otherwise, normalize x = 2^n * 1.f.
 *      Use a 6-bit table look-up: find a 6 bit g that match f to 6.5 bits,
 *      then
 *          log(x) = n*ln2 + log(1.g) + log(1.f/1.g).
 *      Here the leading and trailing values of log(1.g) are obtained from
 *      a size-64 table.
 *      For log(1.f/1.g), let s = (1.f-1.g)/(1.f+1.g), then
 *          log(1.f/1.g) = log((1+s)/(1-s)) = 2s + 2/3 s^3 + 2/5 s^5 +...
 *      Note that |s|<2**-8=0.00390625. We use an odd s-polynomial
 *      approximation to compute log(1.f/1.g):
 *              s*(A1+s^2*(A2+s^2*(A3+s^2*(A4+s^2*(A5+s^2*(A6+s^2*A7))))))
 *      (Precision is 2**-136.91 bits, absolute error)
 *
 * (c). The final result is computed by
 *              (n*ln2_hi+_TBL_logl_hi[j]) +
 *                      ( (n*ln2_lo+_TBL_logl_lo[j]) + s*(A1+...) )
 *
```



```
 * Note.
 *      For ln2_hi and _TBL_logl_hi[j], we force their last 32 bit to be zero
 *      so that n*ln2_hi + _TBL_logl_hi[j] is exact. Here
 *      _TBL_logl_hi[j] + _TBL_logl_lo[j] match log(1+j*2**-6) to 194 bits
 *
 *
 * Special cases:
 *      log(x) is NaN with signal if x < 0 (including -INF) ;
 *      log(+INF) is +INF; log(0) is -INF with signal;
 *      log(NaN) is that NaN with no signal.
 *
 * Constants:
 * The hexadecimal values are the intended ones for the following constants.
 * The decimal values may be used, provided that the compiler will convert
 * from decimal to binary accurately enough to produce the hexadecimal values
 * shown.
 */

#include "libm.h"

extern const long double _TBL_logl_hi[], _TBL_logl_lo[];
static const long double zero = 0.0L,

static const long double
        zero    =   0.0L,
        one = 1.0L,
        two = 2.0L,
        two113 = 10384593717069655257060992658440192.0L,
        ln2hi = 6.931471805599453094172319547495844850203e-0001L,
        ln2lo = 1.667085920830552208890449330400379754169e-0025L,
        A1 = 2.000000000000000000000000000000000000024e+0000L,
        A2 = 6.666666666666666666666666666666091393804e-0001L,
        A3 = 4.000000000000000000000000407167070220671e-0001L,
        A4 = 2.857142857142857142730077490612903681164e-0001L,
        A5 = 2.222222222222242577702836920812882605099e-0001L,
        A6 = 1.818181816435493395985912667105885828356e-0001L,
        A7 = 1.538537835211839751112067512805496931725e-0001L,
        B1 = 6.666666666666666666666666666666961498329e-0001L,
        B2 = 3.999999999999999999999999999000376550423 58e-0001L,
        B3 = 2.857142857142857142857273426428347457918e-0001L,
        B4 = 2.222222222222222221353229049747910109566e-0001L,
        B5 = 1.818181818181821503532559306309070138046e-0001L,
        B6 = 1.538461538453809210486356084587356788556e-0001L,
        B7 = 1.333333344463358756121456892645178795480e-0001L,
        B8 = 1.176460904783899064854645174603360383792e-0001L,
        B9 = 1.057293869956598995326368602518056990746e-0001L;

long double
logl(long double x)
{
logl(long double x) {
        long double f, s, z, qn, h, t;
        int *px = (int *)&x;
        int *pz = (int *)&z;
        int *px = (int *) &x;
        int *pz = (int *) &z;
        int i, j, ix, i0, i1, n;

        /* get long double precision word ordering */
        if (*(int *)&one == 0) {
        if (*(int *) &one == 0) {
                i0 = 3;
                i1 = 0;
        } else {
                i0 = 0;
                i1 = 3;
```

```
        }

        n = 0;
        ix = px[i0];

#endif /* ! codereview */
        if (ix > 0x3ffee0f8) {          /* if x >  31/33 */
                if (ix < 0x3fff1084) {  /* if x < 33/31 */
                        f = x - one;
                        z = f * f;

                        if (((ix - 0x3fff0000) | px[i1] | px[2] | px[1]) == 0)
                        if (((ix - 0x3fff0000) | px[i1] | px[2] | px[1]) == 0) {
                                return (zero);  /* log(1)= +0 */

                        }
                        s = f / (two + f);      /* |s|<2**-8 */
                        z = s * s;
                        return (f - s * (f - z * (B1 + z * (B2 + z * (B3 + z *
                            (B4 + z * (B5 + z * (B6 + z * (B7 + z * (B8 + z *
                            B9)))))))))));
                        return (f - s * (f - z * (B1 + z * (B2 + z * (B3 +
                                z * (B4 + z * (B5 + z * (B6 + z * (B7 +
                                z * (B8 + z * B9)))))))))));
                }

#endif /* ! codereview */
                if (ix >= 0x7fff0000)
                        return (x + x);         /* x is +inf or NaN */

#endif /* ! codereview */
                goto LARGE_N;
        }

#endif /* ! codereview */
        if (ix >= 0x00010000)
                goto LARGE_N;

#endif /* ! codereview */
        i = ix & 0x7fffffff;

#endif /* ! codereview */
        if ((i | px[i1] | px[2] | px[1]) == 0) {
                px[i0] |= 0x80000000;
                return (one / x);       /* log(0.0) = -inf */
        }

#endif /* ! codereview */
        if (ix < 0) {
                if ((unsigned)ix >= 0xffff0000)
                if ((unsigned) ix >= 0xffff0000)
                        return (x - x);         /* x is -inf or NaN */

#endif /* ! codereview */
                return (zero / zero);           /* log(x<0) is NaN  */
        }

#endif /* ! codereview */
        /* subnormal x */
        x *= two113;
        n = -113;
        ix = px[i0];
LARGE_N:
        n += ((ix + 0x200) >> 16) - 0x3fff;
        ix = (ix & 0x0000ffff) | 0x3fff0000;    /* scale x to [1,2] */
        px[i0] = ix;
```



```
        i = ix + 0x200;
        pz[i0] = i & 0xfffffc00;
        pz[i1] = pz[1] = pz[2] = 0;
        s = (x - z) / (x + z);
        j = (i >> 10) & 0x3f;
        z = s * s;
        qn = (long double)n;
        qn = (long double) n;
        t = qn * ln2lo + _TBL_logl_lo[j];
        h = qn * ln2hi + _TBL_logl_hi[j];
        f = t + s * (A1 + z * (A2 + z * (A3 + z * (A4 + z * (A5 + z * (A6 + z *
            A7))))));
        f = t + s * (A1 + z * (A2 + z * (A3 + z * (A4 + z * (A5 +
                z * (A6 + z * A7))))));
        return (h + f);
}
```

_____*unchanged_portion_omitted_*

```
**********************************************************
    6464 Thu Jun  6 20:38:52 2019
new/usr/src/lib/libm/common/Q/longdouble.h
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#ifndef _Q_LONGDOUBLE_H
#define _Q_LONGDOUBLE_H

#include <sys/ieeefp.h>

extern long double __k_cosl(long double, long double);
extern long double __k_lgammal(long double, int *);
extern long double __k_sincosl(long double, long double, long double *);
extern long double __k_sinl(long double, long double);
extern long double __k_tanl(long double, long double, int);
extern long double __poly_libmq(long double, int, const long double *);
extern int __rem_pio2l(long double, long double *);

extern long double acosdl(long double);
extern long double acoshl(long double);
extern long double acosl(long double);
extern long double acospil(long double);
extern long double acospl(long double);
extern long double aintl(long double);
extern long double anintl(long double);
extern long double annuityl(long double, long double);
extern long double asindl(long double);
extern long double asinhl(long double);
extern long double asinl(long double);
extern long double asinpil(long double);
extern long double asinpl(long double);
extern long double atan2dl(long double, long double);
extern long double atan2l(long double, long double);
extern long double atan2pil(long double, long double);
extern long double atandl(long double);
```



```
extern long double atanhl(long double);
extern long double atanl(long double);
extern long double atanpil(long double);
extern long double atanpl(long double);
extern long double cbrtl(long double);
extern long double ceill(long double);
extern long double compoundl(long double, long double);
extern long double copysignl(long double, long double);
extern long double cosdl(long double);
extern long double coshl(long double);
extern long double cosl(long double);
extern long double cospil(long double);
extern long double cospl(long double);
extern long double erfcl(long double);
extern long double erfl(long double);
extern long double exp10l(long double);
extern long double exp2l(long double);
extern long double expl(long double);
extern long double expm1l(long double);
extern long double fabsl(long double);
extern int finitel(long double);
extern long double floorl(long double);
extern long double fmodl(long double, long double);
extern enum fp_class_type fp_classl(long double);
extern long double gammal(long double);
extern long double hypotl(long double, long double);
extern int ilogbl(long double);
extern long double infinityl(void);
extern int irintl(long double);
extern int isinfl(long double);
extern int isnanl(long double);
extern int isnormall(long double);
extern int issubnormall(long double);
extern int iszerol(long double);
extern long double j0l(long double);
extern long double j1l(long double);
extern long double jnl(int, long double);
extern long double lgammal(long double);
extern long double log10l(long double);
extern long double log1pl(long double);
extern long double log2l(long double);
extern long double logbl(long double);
extern long double logl(long double);
extern long double max_normall(void);
extern long double max_subnormall(void);
extern long double min_normall(void);
extern long double min_subnormall(void);
extern long double nextafterl(long double, long double);
extern int nintl(long double);
extern long double pow_li(long double *, int *);
extern long double powl(long double, long double);
extern long double quiet_nanl(long);
extern long double remainderl(long double, long double);
extern long double rintl(long double);
extern long double scalbl(long double, long double);
extern long double scalbnl(long double, int);
extern long double signaling_nanl(long);
extern int signbitl(long double);
extern long double significandl(long double);
extern void sincosdl(long double, long double *, long double *);
extern void sincosl(long double, long double *, long double *);
extern void sincospil(long double, long double *, long double *);
extern void sincospl(long double, long double *, long double *);
extern long double sindl(long double);
extern long double sinhl(long double);
extern long double sinl(long double);
```

```
extern long double sinpil(long double);
extern long double sinpl(long double);
extern long double sqrtl(long double);
extern long double tandl(long double);
extern long double tanhl(long double);
extern long double tanl(long double);
extern long double tanpil(long double);
extern long double tanpl(long double);
extern long double y0l(long double);
extern long double y1l(long double);
extern long double ynl(int, long double);

extern long double q_copysign_(long double *, long double *);
extern long double q_fabs_(long double *);
extern int iq_finite_(long double *);
extern long double q_fmod_(long double *, long double *);
extern enum fp_class_type iq_fp_class_(long double *);
extern int iq_ilogb_(long double *);
extern long double q_infinity_(void);
extern int iq_isinf_(long double *);
extern int iq_isnan_(long double *);
extern int iq_isnormal_(long double *);
extern int iq_issubnormal_(long double *);
extern int iq_iszero_(long double *);
extern long double q_max_normal_(void);
extern long double q_max_subnormal_(void);
extern long double q_min_normal_(void);
extern long double q_min_subnormal_(void);
extern long double q_nextafter_(long double *, long double *);
extern long double q_quiet_nan_(long *);
extern long double q_remainder_(long double *, long double *);
extern long double q_scalbn_(long double *, int *);
extern long double q_signaling_nan_(long *);
extern int iq_signbit_(long double *);

#endif /* _Q_LONGDOUBLE_H */
```

```
**********************************************************
    2728 Thu Jun  6 20:38:52 2019
new/usr/src/lib/libm/common/Q/nextafterl.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __nextafterl = nextafterl

#include "libm.h"
#include <float.h>              /* LDBL_MAX, LDBL_MIN */

#if defined(__sparc)
#define n0      0
#define n1      1
#define n2      2
#define n3      3
#define X86PDNRM1(x)
#define INC(px) { \
                        if (++px[n3] == 0) \
                                if (++px[n2] == 0) \
                                        if (++px[n1] == 0) \
                                                ++px[n0]; \
                }
#define DEC(px) { \
                        if (--px[n3] == 0xffffffff) \
                                if (--px[n2] == 0xffffffff) \
                                        if (--px[n1] == 0xffffffff) \
                                                --px[n0]; \
                }
#elif defined(__x86)
#define n0      2
#define n1      1
#define n2      0
#define n3      0
/*
 * if pseudo-denormal, replace by the equivalent normal
 */
#define X86PDNRM1(x)    if (XBIASED_EXP(x) == 0 && (((int *)&x)[1] & \
```



```
#define X86PDNRM1(x)    if (XBIASED_EXP(x) == 0 && (((int *) &x)[1] & \
                                0x80000000) != 0) \
                                ((int *)&x)[2] |= 1
                                ((int *) &x)[2] |= 1
#define INC(px) { \
                        if (++px[n2] == 0) \
                                if ((++px[n1] & ~0x80000000) == 0) \
                                        px[n1] = 0x80000000, ++px[n0]; \
                }
#define DEC(px) { \
                        if (--px[n2] == 0xffffffff) \
                                if (--px[n1] == 0x7fffffff) \
                                        if ((--px[n0] & 0x7fff) != 0) \
                                                px[n1] |= 0x80000000; \
                }
#endif

long double
nextafterl(long double x, long double y)
{
        int *px = (int *)&x;
        int *py = (int *)&y;
nextafterl(long double x, long double y) {
        int *px = (int *) &x;
        int *py = (int *) &y;

        if (x == y)
                return (y);             /* C99 requirement */
        if (x != x || y != y)
                return (x * y);

        if (ISZEROL(x)) {       /* x == 0.0 */
                px[n0] = py[n0] & XSGNMSK;
                px[n1] = px[n2] = 0;
                px[n3] = 1;
        } else {
                X86PDNRM1(x);
                if ((px[n0] & XSGNMSK) == 0) {  /* x > 0.0 */
                        if (x > y)      /* x > y */
                                DEC(px)
                        else
                                INC(px)
                } else {
                        if (x < y)      /* x < y */
                                DEC(px)
                        else
                                INC(px)
                }
        }
#ifndef lint
        {
                volatile long double dummy;
                int k = XBIASED_EXP(x);

                if (k == 0)
                        dummy = LDBL_MIN * copysignl(LDBL_MIN, x);
                else if (k == 0x7fff)
                        dummy = LDBL_MAX * copysignl(LDBL_MAX, x);
        }
#endif
        return (x);
}
_____unchanged_portion_omitted_
```

```
**********************************************************
    9192 Thu Jun  6 20:38:53 2019
new/usr/src/lib/libm/common/Q/powl.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __powl = powl

#include "libm.h"
#include "xpg6.h"                       /* __xpg6 */
#define _C99SUSv3_pow           _C99SUSv3_pow_treats_Inf_as_an_even_int

#if defined(__sparc)
#define i0                      0
#define i1                      1
#define i2                      2
#define i3                      3

static const long double zero = 0.0L, one = 1.0L, two = 2.0L;

extern const long double _TBL_logl_hi[], _TBL_logl_lo[];
static const long double two113 = 10384593717069655257060992658440192.0L,

static const long double
        two113 = 10384593717069655257060992658440192.0L,
        ln2hi = 6.931471805599453094172319547495844850203e-0001L,
        ln2lo = 1.667085920830552208890449330400379754169e-0025L,
        A2 = 6.666666666666666666666666666666091393804e-0001L,
        A3 = 4.000000000000000000000000407167070220671e-0001L,
        A4 = 2.857142857142857142730077490612903681164e-0001L,
        A5 = 2.222222222222425777028369208128826050 99e-0001L,
        A6 = 1.818181816435493395985912667105885828356e-0001L,
        A7 = 1.538537835211839751112067512805496931725e-0001L,
        B1 = 6.666666666666666666666666666666667787e-0001L,
        B2 = 3.999999999999999999999999999999848524411e-0001L,
        B3 = 2.857142857142857142857142865084581075070e-0001L,
```



```
        B4 = 2.222222222222222222222010781800643808497e-0001L,
        B5 = 1.818181818181818185051442171337036403674e-0001L,
        B6 = 1.538461538461508363540720286292008207673e-0001L,
        B7 = 1.333333333506731842033180638329317108428e-0001L,
        B8 = 1.176469984587418890634302788283946761670e-0001L,
        B9 = 1.053794891561452331722969901564862497132e-0001L;

static long double
logl_x(long double x, long double *w)
{
logl_x(long double x, long double *w) {
        long double f, f1, v, s, z, qn, h, t;
        int *px = (int *)&x;
        int *pz = (int *)&z;
        int *px = (int *) &x;
        int *pz = (int *) &z;
        int i, j, ix, n;

        n = 0;
        ix = px[i0];

#endif /* ! codereview */
        if (ix > 0x3ffef03f && ix < 0x3fff0820) {       /* 65/63 > x > 63/65 */
                f = x - one;
                z = f * f;

#endif /* ! codereview */
                if (((ix - 0x3fff0000) | px[i1] | px[i2] | px[i3]) == 0) {
                        *w = zero;
                        return (zero);  /* log(1)= +0 */
                }

#endif /* ! codereview */
                qn = one / (two + f);
                s = f * qn;             /* |s|<2**-6 */
                v = s * s;
                h = (long double)(2.0 * (double)s);
                f1 = (long double)((double)f);
                t = ((two * (f - h) - h * f1) - h * (f - f1)) * qn + s * (v *
                    (B1 + v * (B2 + v * (B3 + v * (B4 + v * (B5 + v * (B6 + v *
                    (B7 + v * (B8 + v * B9))))))))));
                s = (long double)((double)(h + t));
                h = (long double) (2.0 * (double) s);
                f1 = (long double) ((double) f);
                t = ((two * (f - h) - h * f1) - h * (f - f1)) * qn +
                        s * (v * (B1 + v * (B2 + v * (B3 + v * (B4 +
                        v * (B5 + v * (B6 + v * (B7 + v * (B8 + v * B9))))))))));
                s = (long double) ((double) (h + t));
                *w = t - (s - h);
                return (s);
        }

#endif /* ! codereview */
        if (ix < 0x00010000) {          /* subnormal x */
                x *= two113;
                n = -113;
                ix = px[i0];
        }

#endif /* ! codereview */
        /* LARGE_N */
        n += ((ix + 0x200) >> 16) - 0x3fff;
        ix = (ix & 0x0000ffff) | 0x3fff0000;    /* scale x to [1,2] */
        px[i0] = ix;
        i = ix + 0x200;
        pz[i0] = i & 0xffffc00;
```

```
115         pz[i1] = pz[i2] = pz[i3] = 0;
116         qn = one / (x + z);
117         f = x - z;
118         s = f * qn;
119         f1 = (long double)((double)f);
120         h = (long double)(2.0 * (double)s);
 66         f1 = (long double) ((double) f);
 67         h = (long double) (2.0 * (double) s);
121         t = qn * ((two * (f - z * h) - h * f1) - h * (f - f1));
122         j = (i >> 10) & 0x3f;
123         v = s * s;
124         qn = (long double)n;
 71         qn = (long double) n;
125         t += qn * ln2lo + _TBL_logl_lo[j];
126         t += s * (v * (A2 + v * (A3 + v * (A4 + v * (A5 + v * (A6 + v *
127             A7))))));
 73         t += s * (v * (A2 + v * (A3 + v * (A4 + v * (A5 + v * (A6 +
 74                 v * A7))))));
128         v = qn * ln2hi + _TBL_logl_hi[j];
129         s = h + v;
130         t += (h - (s - v));
131         z = (long double)((double)(s + t));
 78         z = (long double) ((double) (s + t));
132         *w = t - (z - s);
133         return (z);
134 }

136 extern const long double _TBL_expl_hi[], _TBL_expl_lo[];
137 static const long double
138         invln2_32 = 4.616624130844682903551758979206054839765e+1L,
139         ln2_32hi = 2.166084939249829091928849858592451515688e-2L,
140         ln2_32lo = 5.209643502595475652782654157501186731779e-27L,
141         ln2_64 = 1.083042469624914545964425189778400898568e-2L;

143 long double
144 powl(long double x, long double y)
145 {
 91 powl(long double x, long double y) {
146         long double z, ax;
147         long double y1, y2, w1, w2;
148         int sbx, sby, j, k, yisint, m;
149         int hx, lx, hy, ly, ahx, ahy;
150         int *pz = (int *)&z;
151         int *px = (int *)&x;
152         int *py = (int *)&y;
 96         int *pz = (int *) &z;
 97         int *px = (int *) &x;
 98         int *py = (int *) &y;

154         hx = px[i0];
155         lx = px[i1] | px[i2] | px[i3];
156         hy = py[i0];
157         ly = py[i1] | py[i2] | py[i3];
158         ahx = hx & ~0x80000000;
159         ahy = hy & ~0x80000000;

161         if ((ahy | ly) == 0)
162                 return (one);           /* x**+-0 = 1 */
163         else if (hx == 0x3fff0000 && lx == 0 && (__xpg6 & _C99SUSv3_pow) != 0)
109         else if (hx == 0x3fff0000 && lx == 0 &&
110                 (__xpg6 & _C99SUSv3_pow) != 0)
164                 return (one);           /* C99: 1**anything = 1 */
165         else if (ahx > 0x7fff0000 || (ahx == 0x7fff0000 && lx != 0) || ahy >
166             0x7fff0000 || (ahy == 0x7fff0000 && ly != 0))
112         else if (ahx > 0x7fff0000 || (ahx == 0x7fff0000 && lx != 0) ||
113                 ahy > 0x7fff0000 || (ahy == 0x7fff0000 && ly != 0))
```



```
167                 return (x + y);         /* +-NaN return x+y */

169 #endif /* ! codereview */
170         /* includes Sun: 1**NaN = NaN */
171         sbx = (unsigned)hx >> 31;
172         sby = (unsigned)hy >> 31;
115         sbx = (unsigned) hx >> 31;
116         sby = (unsigned) hy >> 31;
173         ax = fabsl(x);

175 #endif /* ! codereview */
176         /*
177          * determine if y is an odd int when x < 0
178          * yisint = 0 ... y is not an integer
179          * yisint = 1 ... y is an odd int
180          * yisint = 2 ... y is an even int
181          */
182         yisint = 0;

184 #endif /* ! codereview */
185         if (sbx) {
186                 if (ahy >= 0x40700000) { /* if |y|>=2**113 */
118                 if (ahy >= 0x40700000)  /* if |y|>=2**113 */
187                         yisint = 2;                     /* even integer y */
188                 } else if (ahy >= 0x3fff0000) {
120                 else if (ahy >= 0x3fff0000) {
189                         k = (ahy >> 16) - 0x3fff;       /* exponent */

191 #endif /* ! codereview */
192                         if (k > 80) {
193                                 j = ((unsigned)py[i3]) >> (112 - k);

122                                 j = ((unsigned) py[i3]) >> (112 - k);
195                                 if ((j << (112 - k)) == py[i3])
196                                         yisint = 2 - (j & 1);
197                         } else if (k > 48) {
198                                 j = ((unsigned)py[i2]) >> (80 - k);

126                                 j = ((unsigned) py[i2]) >> (80 - k);
200                                 if ((j << (80 - k)) == py[i2])
201                                         yisint = 2 - (j & 1);
202                         } else if (k > 16) {
203                                 j = ((unsigned)py[i1]) >> (48 - k);

130                                 j = ((unsigned) py[i1]) >> (48 - k);
205                                 if ((j << (48 - k)) == py[i1])
206                                         yisint = 2 - (j & 1);
207                         } else if (ly == 0) {
208                                 j = ahy >> (16 - k);

210 #endif /* ! codereview */
211                                 if ((j << (16 - k)) == ahy)
212                                         yisint = 2 - (j & 1);
213                         }
214                 }
215         }

217         /* special value of y */
218         if (ly == 0) {
219                 if (ahy == 0x7fff0000) {        /* y is +-inf */
220                         if (((ahx - 0x3fff0000) | lx) == 0) {
221                                 if ((__xpg6 & _C99SUSv3_pow) != 0)
222                                         return (one);
223                                 /* C99: (-1)**+-inf = 1 */
224                                 else
225                                         return (y - y);
```

```
#endif /* ! codereview */
                                /* Sun: (+-1)**+-inf = NaN */
                        } else if (ahx >= 0x3fff0000) {
                        } else if (ahx >= 0x3fff0000)
                                /* (|x|>1)**+,-inf = inf,0 */
                                return (sby == 0 ? y : zero);
                        } else {                /* (|x|<1)**-,+inf = inf,0 */
                        else                    /* (|x|<1)**-,+inf = inf,0 */
                                return (sby != 0 ? -y : zero);
                        }
#endif /* ! codereview */
                } else if (ahy == 0x3fff0000) { /* y is +-1 */
                        if (sby != 0)
                                return (one / x);
                        else
                                return (x);
                } else if (hy == 0x40000000) {  /* y is 2 */
                } else if (hy == 0x40000000)    /* y is 2 */
                        return (x * x);
                } else if (hy == 0x3ffe0000) {  /* y is 0.5 */
                else if (hy == 0x3ffe0000) {    /* y is 0.5 */
                        if (!((ahx | lx) == 0 || ((ahx - 0x7fff0000) | lx) ==
                            0))
                                return (sqrtl(x));
                }
        }

        /* special value of x */
        if (lx == 0) {
                if (ahx == 0x7fff0000 || ahx == 0 || ahx == 0x3fff0000) {
                        /* x is +-0,+-inf,+-1 */
                        z = ax;

#endif /* ! codereview */
                        if (sby == 1)
                                z = one / z;    /* z = 1/|x| if y is negative */

#endif /* ! codereview */
                        if (sbx == 1) {
                                if (ahx == 0x3fff0000 && yisint == 0)
                                        z = zero / zero;
                                /* (-1)**non-int is NaN */
                                else if (yisint == 1)
                                        z = -z; /* (x<0)**odd = -(|x|**odd) */
                        }

#endif /* ! codereview */
                        return (z);
                }
        }

        /* (x<0)**(non-int) is NaN */
        if (sbx == 1 && yisint == 0)
                return (zero / zero);   /* should be volatile */

        /*
         * Now ax is finite, y is finite
         * first compute log(ax) = w1+w2, with 53 bits w1
         */
        /* Now ax is finite, y is finite */
        /* first compute log(ax) = w1+w2, with 53 bits w1 */
        w1 = logl_x(ax, &w2);

        /* split up y into y1+y2 and compute (y1+y2)*(w1+w2) */
        if (ly == 0 || ahy >= 0x43fe0000) {
```



```
                y1 = y * w1;
                y2 = y * w2;
        } else {
                y1 = (long double)((double)y);
                y1 = (long double) ((double) y);
                y2 = (y - y1) * w1 + y * w2;
                y1 *= w1;
        }

#endif /* ! codereview */
        z = y1 + y2;
        j = pz[i0];

        if ((unsigned)j >= 0xffff0000) {        /* NaN or -inf */
        if ((unsigned) j >= 0xffff0000) {               /* NaN or -inf */
                if (sbx == 1 && yisint == 1)
                        return (one / z);
                else
                        return (-one / z);
        } else if ((j & ~0x80000000) < 0x3fc30000) {    /* |x|<2^-60 */
                if (sbx == 1 && yisint == 1)
                        return (-one - z);
                else
                        return (one + z);
        } else if (j > 0) {
                if (j > 0x400d0000) {
                        if (sbx == 1 && yisint == 1)
                                return (scalbnl(-one, 20000));
                        else
                                return (scalbnl(one, 20000));
                }

                k = (int)(invln2_32 * (z + ln2_64));
                k = (int) (invln2_32 * (z + ln2_64));
        } else {
                if ((unsigned)j > 0xc00d0000) {
                if ((unsigned) j > 0xc00d0000) {
                        if (sbx == 1 && yisint == 1)
                                return (scalbnl(-one, -20000));
                        else
                                return (scalbnl(one, -20000));
                }

                k = (int)(invln2_32 * (z - ln2_64));
                k = (int) (invln2_32 * (z - ln2_64));
        }

#endif /* ! codereview */
        j = k & 0x1f;
        m = k >> 5;
        {
                /* rational approximation coeffs for [-(ln2)/64,(ln2)/64] */
                long double t1 = 1.666666666666666666666666666660876387437e-1L,
                long double
                        t1 = 1.666666666666666666666666666660876387437e-1L,
                    t2 = -2.777777777777777777777707812093173478756e-3L,
                    t3 = 6.613756613756613482074280932874221202424e-5L,
                    t4 = -1.653439153392139954169609822742235851120e-6L,
                    t5 = 4.175314851769539751387852116610973796053e-8L;
                long double t = (long double)k;
                long double t = (long double) k;

                w1 = (y2 - (t * ln2_32hi - y1)) - t * ln2_32lo;
                t = w1 * w1;
                w2 = (w1 - t * (t1 + t * (t2 + t * (t3 + t * (t4 + t * t5))))) -
                    two;
```

```
                z = _TBL_expl_hi[j] - ((_TBL_expl_hi[j] * (w1 + w1)) / w2 -
                    _TBL_expl_lo[j]);
        }
        j = m + (pz[i0] >> 16);

        if (j && (unsigned)j < 0x7fff)
        if (j && (unsigned) j < 0x7fff)
                pz[i0] += m << 16;
        else
                z = scalbnl(z, m);

        if (sbx == 1 && yisint == 1)
                z = -z;                 /* (-ve)**(odd int) */

#endif /* ! codereview */
        return (z);
}
#else
#error Unsupported Architecture
#endif /* defined(__sparc) */
```

```
**********************************************************
    2229 Thu Jun  6 20:38:53 2019
new/usr/src/lib/libm/common/Q/remainderl.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __remainderl = remainderl

#include "libm.h"
#include "longdouble.h"

/*
 * remainderl(x,p)
 *      returns  x REM p  =  x - [x/p]*p as if in infinite
 *      precise arithmetic, where [x/p] is the (inifinite bit)
 *      integer nearest x/p (in half way case choose the even one).
 * Method :
 *      Based on fmodl() return x-[x/p]chopped*p exactly.
 */

#define HFMAX           5.948657476786158825428796633140035080982e+4931L
#define DBMIN           6.724206286224187012525355634643505205196e-4932L

static const long double zero = 0.0L,
static const long double
        zero = 0.0L,
        half = 0.5L,
        hfmax = HFMAX,  /* half of the maximum number */
        dbmin = DBMIN;  /* double of the minimum (normal) number */

long double
remainderl(long double x, long double p)
{
remainderl(long double x, long double p) {
        long double hp;
        int sx;
```



```
        if (isnanl(p))
                return (x + p);

#endif /* ! codereview */
        if (!finitel(x))
                return (x - x);

#endif /* ! codereview */
        p = fabsl(p);

#endif /* ! codereview */
        if (p <= hfmax)
                x = fmodl(x, p + p);

#endif /* ! codereview */
        sx = signbitl(x);
        x = fabsl(x);

#endif /* ! codereview */
        if (p < dbmin) {
                if (x + x > p) {
                        if (x == p)
                                x = zero;
                        else
                                x -= p;         /* avoid x-x=-0 in RM mode */

#endif /* ! codereview */
                        if (x + x >= p)
                                x -= p;
                }
        } else {
                hp = half * p;

#endif /* ! codereview */
                if (x > hp) {
                        if (x == p)
                                x = zero;
                        else
                                x -= p;         /* avoid x-x=-0 in RM mode */

#endif /* ! codereview */
                        if (x >= hp)
                                x -= p;
                }
        }

#endif /* ! codereview */
        return (sx == 0 ? x : -x);
}
```

```
**********************************************************
    2139 Thu Jun  6 20:38:54 2019
new/usr/src/lib/libm/common/Q/rintl.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __rintl = rintl

/*
 * rintl(long double x) return x rounded to integral according to
 * the prevailing rounding direction
 *
 * NOTE: aintl(x), anintl(x), ceill(x), floorl(x), and rintl(x) return result
 * with the same sign as x's, including 0.0L.
 */

#include "libm.h"
#include "longdouble.h"

extern enum fp_precision_type __swapRP(enum fp_precision_type);

static const double one = 1.0;
static const long double qzero = 0.0L;

long double
rintl(long double x)
{
rintl(long double x) {
        enum fp_precision_type rp;
        long double t, w, two112;
        int *pt = (int *)&two112;
        int *pt = (int *) &two112;

        if (!finitel(x))
                return (x + x);
```



```
        if (*(int *)&one != 0) {        /* set two112 = 2^112 */
        if (*(int *) &one != 0) {       /* set two112 = 2^112 */
                pt[0] = 0x406f0000;
                pt[1] = pt[2] = pt[3] = 0;
        } else {
                pt[3] = 0x406f0000;
                pt[0] = pt[1] = pt[2] = 0;
        }

        if (fabsl(x) >= two112)
                return (x);                     /* already an integer */

#endif /* ! codereview */
        t = copysignl(two112, x);
        rp = __swapRP(fp_extended); /* make sure precision is long double */
        w = x + t;                  /* x+sign(x)*2^112 rounded to integer */
        (void) __swapRP(rp);        /* restore precision mode */

#endif /* ! codereview */
        if (w == t)
                return (copysignl(qzero, x));   /* x rounded to zero */
        else
                return (w - t);
}
```

```
**********************************************************
    2850 Thu Jun  6 20:38:54 2019
new/usr/src/lib/libm/common/Q/rndintl.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

#endif /* ! codereview */
/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2005 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

/*
 * aintl(x)     return x chopped to integral value
 * anintl(x)    return sign(x)*(|x|+0.5) chopped to integral value
 * irintl(x)    return rint(x) in integer format
 * nintl(x)     return anint(x) in integer format
 *
 * NOTE: aintl(x), anintl(x), ceill(x), floorl(x), and rintl(x) return result
 * with the same sign as x's,  including 0.0.
 */

#include "libm.h"
#include "longdouble.h"

extern enum fp_direction_type __swapRD(enum fp_direction_type);

static const long double qone = 1.0L, qhalf = 0.5L, qmhalf = -0.5L;

long double
aintl(long double x)
{
aintl(long double x) {
        long double t, w;

        if (!finitel(x))
                return (x + x);

#endif /* ! codereview */
        w = fabsl(x);
        t = rintl(w);
```



```
#endif /* ! codereview */
        if (t <= w)
                return (copysignl(t, x)); /* NaN or already aint(|x|) */
        else                              /* |t|>|x| case */
                return (copysignl(t - qone, x)); /* |t-1|*sign(x) */
}

long double
anintl(long double x)
{
anintl(long double x) {
        long double t, w, z;

        if (!finitel(x))
                return (x + x);

#endif /* ! codereview */
        w = fabsl(x);
        t = rintl(w);

#endif /* ! codereview */
        if (t == w)
                return (copysignl(t, x));

#endif /* ! codereview */
        z = t - w;

#endif /* ! codereview */
        if (z > qhalf)
                t = t - qone;
        else if (z <= qmhalf)
                t = t + qone;

#endif /* ! codereview */
        return (copysignl(t, x));
}

int
irintl(long double x)
{
irintl(long double x) {
        enum fp_direction_type rd;

        rd = __swapRD(fp_nearest);
        (void) __swapRD(rd);            /* restore Rounding Direction */

#endif /* ! codereview */
        switch (rd) {
        case fp_nearest:

#endif /* ! codereview */
                if (x < 2147483647.5L && x >= -2147483648.5L)
                        return ((int)rintl(x));

#endif /* ! codereview */
                break;
        case fp_tozero:

#endif /* ! codereview */
                if (x < 2147483648.0L && x > -2147483649.0L)
                        return ((int)rintl(x));

#endif /* ! codereview */
                break;
        case fp_positive:
```

```
#endif /* ! codereview */
                if (x <= 2147483647.0L && x > -2147483649.0L)
                        return ((int)rintl(x));

#endif /* ! codereview */
                break;
        case fp_negative:

#endif /* ! codereview */
                if (x < 2147483648.0L && x >= -2147483648.0L)
                        return ((int)rintl(x));

#endif /* ! codereview */
                break;
        }

#endif /* ! codereview */
        return ((int)copysignl(1.0e100L, x));
}

int
nintl(long double x)
{
nintl(long double x) {
        if ((x < 2147483647.5L) && (x > -2147483648.5L))
                return ((int)anintl(x));
        else
                return ((int)copysignl(1.0e100L, x));
}
```

*_____unchanged_portion_omitted_*

```
**********************************************************
    1660 Thu Jun  6 20:38:55 2019
new/usr/src/lib/libm/common/Q/scalbl.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __scalbl = scalbl

/*
 * scalbl(x,n): return x * 2^n by manipulating exponent.
 */

#include "libm.h"

#define n0      0

long double
scalbl(long double x, long double fn)
{
        int *py = (int *)&fn, n;
scalbl(long double x, long double fn) {
        int *py = (int *) &fn, n;
        long double z;

        if (isnanl(x) || isnanl(fn))
                return (x * fn);

        /*
         * fn is inf or NaN
         */
        if ((py[n0] & 0x7fff0000) == 0x7fff0000) {
                if ((py[n0] & 0x80000000) != 0)
                        return (x / (-fn));
                else
                        return (x * fn);
        }
```



```
#endif /* ! codereview */
        if (rintl(fn) != fn)
                return ((fn - fn) / (fn - fn));

        if (fn > 65000.0L) {
        if (fn > 65000.0L)
                z = scalbnl(x, 65000);
        } else if (-fn > 65000.0L) {
        else if (-fn > 65000.0L)
                z = scalbnl(x, -65000);
        } else {
                n = (int)fn;
        else {
                n = (int) fn;
                z = scalbnl(x, n);
        }

#endif /* ! codereview */
        return (z);
}
```

```
**********************************************************
    2518 Thu Jun  6 20:38:55 2019
new/usr/src/lib/libm/common/Q/scalbnl.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __scalbnl = scalbnl

#include "libm.h"
#include <float.h>                      /* LDBL_MAX, LDBL_MIN */
#include <stdlib.h>                     /* abs */

#if defined(__sparc)
#define XSET_EXP(k, x)          ((int *)&x)[0] = (((int *)&x)[0] & \
        ~0x7fff0000) | (k << 16)
#define XSET_EXP(k, x)  ((int *) &x)[0] = (((int *) &x)[0] & ~0x7fff0000) | \
                                (k << 16)
#define ISINFNANL(k, x)         (k == 0x7fff)
#define XTWOT_OFFSET            113

/* 2^113 */
static const long double xtwot = 10384593717069655257060992658440192.0L;
/* 2^-114 */
static const long double twomtm1 = 4.81482486096808963263994485646231829 6E-35L;
static const long double xtwot = 10384593717069655257060992658440192.0L,
                                                        /* 2^113 */
        twomtm1 = 4.81482486096808963263994485646231829 6E-35L;  /* 2^-114 */
#elif defined(__x86)
#define XSET_EXP(k, x)          ((int *)&x)[2] = (((int *)&x)[2] & ~0x7fff) | k
#define XSET_EXP(k, x)  ((int *) &x)[2] = (((int *) &x)[2] & ~0x7fff) | k
#if defined(HANDLE_UNSUPPORTED)
#define ISINFNANL(k, x)         (k == 0x7fff || k != 0 && (((int *)&x)[1] & \
        0x80000000) == 0)
#define ISINFNANL(k, x) (k == 0x7fff || k != 0 && \
                                (((int *) &x)[1] & 0x80000000) == 0)
#else
```



```
#define ISINFNANL(k, x)         (k == 0x7fff)
#endif
#define XTWOT_OFFSET            64

#endif /* ! codereview */
static const long double xtwot = 18446744073709551616.0L,       /* 2^64 */
        twomtm1 = 2.7105054312137610850186E-20L;                /* 2^-65 */
#endif

long double
scalbnl(long double x, int n)
{
scalbnl(long double x, int n) {
        int k = XBIASED_EXP(x);

        if (ISINFNANL(k, x))
                return (x + x);

#endif /* ! codereview */
        if (ISZEROL(x) || n == 0)
                return (x);

#endif /* ! codereview */
        if (k == 0) {
                x *= xtwot;
                k = XBIASED_EXP(x) - XTWOT_OFFSET;
        }

        if ((unsigned)abs(n) >= 131072) /* cast to unsigned for -2^31 */
        if ((unsigned) abs(n) >= 131072)        /* cast to unsigned for -2^31 */
                n >>= 1;                /* avoid subsequent integer overflow */

#endif /* ! codereview */
        k += n;

#endif /* ! codereview */
        if (k > 0x7ffe)
                return (LDBL_MAX * copysignl(LDBL_MAX, x));

#endif /* ! codereview */
        if (k <= -XTWOT_OFFSET - 1)
                return (LDBL_MIN * copysignl(LDBL_MIN, x));

#endif /* ! codereview */
        if (k > 0) {
                XSET_EXP(k, x);
                return (x);
        }

#endif /* ! codereview */
        k += XTWOT_OFFSET + 1;
        XSET_EXP(k, x);
        return (x * twomtm1);
}
```

```
**********************************************************
    1071 Thu Jun  6 20:38:56 2019
new/usr/src/lib/libm/common/Q/signgaml.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __signgaml = signgaml

#include "libm.h"

int signgaml = 0;
```

```
**********************************************************
    1227 Thu Jun  6 20:38:56 2019
new/usr/src/lib/libm/common/Q/significandl.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __significandl = significandl

#include "libm.h"

long double
significandl(long double x)
{
significandl(long double x) {
        if (ISZEROL(x) || XBIASED_EXP(x) == 0x7fff)     /* 0/+-Inf/NaN */
                return (x + x);
        else
                return (scalbnl(x, -ilogbl(x)));
}
______unchanged_portion_omitted_
```

```
**********************************************************
    2719 Thu Jun  6 20:38:57 2019
new/usr/src/lib/libm/common/Q/sincosl.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

/*
 * sincosl(x)
 * Table look-up algorithm by K.C. Ng, November, 1989.
 *
 * kernel function:
 *      __k_sincosl     ... sin and cos function on [-pi/4,pi/4]
 *      __rem_pio2l     ... argument reduction routine
 *
 * Method.
 *      Let S and C denote the sin and cos respectively on [-PI/4, +PI/4].
 *      1. Assume the argument x is reduced to y1+y2 = x-k*pi/2 in
 *         [-pi/2 , +pi/2], and let n = k mod 4.
 *      2. Let S=S(y1+y2), C=C(y1+y2). Depending on n, we have
 *
 *          n        sin(x)      cos(x)        tan(x)
 *     ----------------------------------------------------------
 *          0          S           C             S/C
 *          1          C          -S            -C/S
 *          2         -S          -C             S/C
 *          3         -C           S            -C/S
 *     ----------------------------------------------------------
 *
 * Special cases:
 *      Let trig be any of sin, cos, or tan.
 *      trig(+-INF)  is NaN, with signals;
 *      trig(NaN)    is that NaN;
 *
 * Accuracy:
 *      computer TRIG(x) returns trig(x) nearly rounded.
 */
```



```
#pragma weak __sincosl = sincosl

#include "libm.h"
#include "longdouble.h"

void
sincosl(long double x, long double *s, long double *c)
{
sincosl(long double x, long double *s, long double *c) {
        long double y[2], z = 0.0L;
        int n, ix;

        ix = *(int *)&x;                /* High word of x */
        ix = *(int *) &x;       /* High word of x */

        /* |x| ~< pi/4 */
        ix &= 0x7fffffff;

        if (ix <= 0x3ffe9220) {
        if (ix <= 0x3ffe9220)
                *s = __k_sincosl(x, z, c);
        } else if (ix >= 0x7fff0000) {
        else if (ix >= 0x7fff0000)
                *s = *c = x - x;        /* trig(Inf or NaN) is NaN */
        } else {                        /* argument reduction needed */
        else {                          /* argument reduction needed */
                n = __rem_pio2l(x, y);

#endif /* ! codereview */
                switch (n & 3) {
                case 0:
                        *s = __k_sincosl(y[0], y[1], c);
                        break;
                case 1:
                        *c = -__k_sincosl(y[0], y[1], s);
                        break;
                case 2:
                        *s = -__k_sincosl(y[0], y[1], c);
                        *c = -*c;
                        break;
                case 3:
                        *c = __k_sincosl(y[0], y[1], s);
                        *s = -*s;
                        break;
                }
        }
}
```

```
**********************************************************
    6003 Thu Jun  6 20:38:57 2019
new/usr/src/lib/libm/common/Q/sincospil.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

/*
 * void sincospil(long double x, long double *s, long double *c)
 * *s = sinl(pi*x); *c = cosl(pi*x);
 *
 * Algorithm, 10/17/2002, K.C. Ng
 * ------------------------------
 * Let y = |4x|, z = floor(y), and n = (int)(z mod 8.0) (displayed in binary).
 *      1. If y == z, then x is a multiple of pi/4. Return the following values:
 *             ----------------------------------------------------
 *               n  x mod 2    sin(x*pi)    cos(x*pi)   tan(x*pi)
 *             ----------------------------------------------------
 *              000  0.00       +0 ___       +1 ___       +0
 *              001  0.25       +\/0.5       +\/0.5       +1
 *              010  0.50       +1 ___       +0 ___       +inf
 *              011  0.75       +\/0.5       -\/0.5       -1
 *              100  1.00       -0 ___       -1 ___       +0
 *              101  1.25       -\/0.5       -\/0.5       +1
 *              110  1.50       -1 ___       -0 ___       +inf
 *              111  1.75       -\/0.5       +\/0.5       -1
 *             ----------------------------------------------------
 *      2. Otherwise,
 *             ----------------------------------------------------
 *               n     t        sin(x*pi)    cos(x*pi)   tan(x*pi)
 *             ----------------------------------------------------
 *              000  (y-z)/4     sinpi(t)     cospi(t)    tanpi(t)
 *              001  (z+1-y)/4   cospi(t)     sinpi(t)    1/tanpi(t)
 *              010  (y-z)/4     cospi(t)    -sinpi(t)   -1/tanpi(t)
 *              011  (z+1-y)/4   sinpi(t)    -cospi(t)   -tanpi(t)
 *              100  (y-z)/4    -sinpi(t)    -cospi(t)    tanpi(t)
 *              101  (z+1-y)/4  -cospi(t)    -sinpi(t)    1/tanpi(t)
```



```
 *              110  (y-z)/4    -cospi(t)     sinpi(t)   -1/tanpi(t)
 *              111  (z+1-y)/4  -sinpi(t)     cospi(t)   -tanpi(t)
 *             ----------------------------------------------------
 *
 * NOTE. This program compute sinpi/cospi(t<0.25) by __k_sin/cos(pi*t, 0.0).
 * This will return a result with error slightly more than one ulp (but less
 * than 2 ulp). If one wants accurate result,  one may break up pi*t in
 * high (tpi_h) and low (tpi_l) parts and call __k_sin/cos(tip_h, tip_lo)
 * instead.
 */

#include "libm.h"
#include "longdouble.h"

#define I(q, m)                         ((int *)&(q))[m]
#define U(q, m)                         ((unsigned *)&(q))[m]
#define I(q, m) ((int *) &(q))[m]
#define U(q, m) ((unsigned *) &(q))[m]
#if defined(__LITTLE_ENDIAN) || defined(__x86)
#define LDBL_MOST_SIGNIF_I(ld)          ((I(ld, 2) << 16) | (0xffff & (I(ld, \
        1) >> 15)))
#define LDBL_MOST_SIGNIF_I(ld)  ((I(ld, 2) << 16) | (0xffff & (I(ld, 1) >> 15)))
#define LDBL_LEAST_SIGNIF_U(ld)         U(ld, 0)
#define PREC                            64
#define PRECM1                          63
#define PRECM2                          62

#endif /* ! codereview */
static const long double twoPRECM2 = 9.22337203685477580800000000000000e+18L;
#else
#define LDBL_MOST_SIGNIF_I(ld)          I(ld, 0)
#define LDBL_LEAST_SIGNIF_U(ld)         U(ld, sizeof (long double) / \
        sizeof (int) - 1)
#define LDBL_LEAST_SIGNIF_U(ld) U(ld, sizeof (long double) / sizeof (int) - 1)
#define PREC                            113
#define PRECM1                          112
#define PRECM2                          111

#endif /* ! codereview */
static const long double twoPRECM2 = 5.19229685853482762853049632922009 6e+33L;
#endif

static const long double zero = 0.0L,
        quater = 0.25L,
        one = 1.0L,
        pi = 3.14159265358979323846264338327950288419 7e+0000L,
        sqrth = 0.70710678118654752440084436210484903928483593768847 4,
        tiny = 1.0e-100;
static const long double
zero    = 0.0L,
quater  = 0.25L,
one     = 1.0L,
pi      = 3.14159265358979323846264338327950288419 7e+0000L,
sqrth   = 0.70710678118654752440084436210484903928483593768847 4,
tiny    = 1.0e-100;

void
sincospil(long double x, long double *s, long double *c)
{
sincospil(long double x, long double *s, long double *c) {
        long double y, z, t;
        int hx, n, k;
        unsigned lx;

        hx = LDBL_MOST_SIGNIF_I(x);
        lx = LDBL_LEAST_SIGNIF_U(x);
```

```
        k = ((hx & 0x7fff0000) >> 16) - 0x3fff;

#endif /* ! codereview */
        if (k >= PRECM2) {              /* |x| >= 2**(Prec-2) */
                if (k >= 16384) {
                        *s = *c = x - x;
                } else {
                        if (k >= PREC) {
                                *s = zero;
                                *c = one;
                        } else if (k == PRECM1) {
                                if ((lx & 1) == 0) {
                                        *s = zero;
                                        *c = one;
                                } else {
                                        *s = -zero;
                                        *c = -one;
                                }
                        } else {        /* k = Prec - 2 */
                                if ((lx & 1) == 0) {
                                        *s = zero;
                                        *c = one;
                                } else {
                                        *s = one;
                                        *c = zero;
                                }

#endif /* ! codereview */
                                if ((lx & 2) != 0) {
                                        *s = -*s;
                                        *c = -*c;
                                }
                        }
                }
        } else if (k < -2) {            /* |x| < 0.25 */
        } else if (k < -2)      /* |x| < 0.25 */
                *s = __k_sincosl(pi * fabsl(x), zero, c);
        } else {
        else {
                /* y = |4x|, z = floor(y), and n = (int)(z mod 8.0) */
                y = 4.0L * fabsl(x);

#endif /* ! codereview */
                if (k < PRECM2) {
                        z = y + twoPRECM2;
                        n = LDBL_LEAST_SIGNIF_U(z) & 7; /* 3 LSb of z */
                        t = z - twoPRECM2;
                        k = 0;

                        if (t == y) {
                        if (t == y)
                                k = 1;
                        } else if (t > y) {
                        else if (t > y) {
                                n -= 1;
                                t = quater + (y - t) * quater;
                        } else {
                        } else
                                t = (y - t) * quater;
                        }
#endif /* ! codereview */
                } else {                                /* k = Prec-3 */
                        n = LDBL_LEAST_SIGNIF_U(y) & 7; /* 3 LSb of z */
                        k = 1;
                }
```



```
#endif /* ! codereview */
                if (k) {                        /* x = N/4 */
                        if ((n & 1) != 0) {
                        if ((n & 1) != 0)
                                *s = *c = sqrth + tiny;
                        } else if ((n & 2) == 0) {
                        else
                                if ((n & 2) == 0) {
                                *s = zero;
                                *c = one;
                        } else {
                                *s = one;
                                *c = zero;
                        }

#endif /* ! codereview */
                        if ((n & 4) != 0)
                                *s = -*s;

#endif /* ! codereview */
                        if (((n + 1) & 4) != 0)
                                *c = -*c;
                } else {
                        if ((n & 1) != 0)
                                t = quater - t;

#endif /* ! codereview */
                        if (((n + (n & 1)) & 2) == 0)
                                *s = __k_sincosl(pi * t, zero, c);
                        else
                                *c = __k_sincosl(pi * t, zero, s);

#endif /* ! codereview */
                        if ((n & 4) != 0)
                                *s = -*s;

#endif /* ! codereview */
                        if (((n + 2) & 4) != 0)
                                *c = -*c;
                }
        }

#endif /* ! codereview */
        if (hx < 0)
                *s = -*s;
}

#endif /* ! codereview */
#undef U
#undef LDBL_LEAST_SIGNIF_U
#undef I
#undef LDBL_MOST_SIGNIF_I
```

```
**********************************************************
    2395 Thu Jun  6 20:38:58 2019
new/usr/src/lib/libm/common/Q/sinhl.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __sinhl = sinhl

#include "libm.h"
#include "longdouble.h"

/*
 * sinhl(X)
 * RETURN THE HYPERBOLIC SINE OF X
 *
 * Method :
 *      1. reduce x to non-negative by sinhl(-x) = - sinhl(x).
 *      2.
 *
 *                                              expm1l(x) + expm1l(x)/(expm1l(x)+1)
 *      0 <= x <= lnovft        : sinhl(x) := -------------------------------
 *                                                          2
 *
 *     lnovft <= x <  INF       : sinhl(x) := expl(x-MEP1*ln2)*2**ME
 *
 * here
 *      lnovft:         logrithm of the overflow threshold
 *                      = MEP1*ln2 chopped to machine precision.
 *      ME              maximum exponent
 *      MEP1            maximum exponent plus 1
 *
 * Special cases:
 *      sinhl(x) is x if x is +INF, -INF, or NaN.
 *      only sinhl(0)=0 is exact for finite argument.
 *
 */
```



```
#define ME              16383
#define MEP1            16384
#define LNOVFT          1.13565234062941439494919310779707634284 5e+4L
/* last 32 bits of LN2HI is zero */
                /* last 32 bits of LN2HI is zero */
#define LN2HI           6.931471805599453094172319547495844850203e-0001L
#define LN2LO           1.667085920830552208890449330400379754169e-0025L

static const long double half = 0.5L,
static const long double
        half    = 0.5L,
        one = 1.0L,
        ln2hi = LN2HI,
        ln2lo = LN2LO,
        lnovftL = LNOVFT;

long double
sinhl(long double x)
{
sinhl(long double x) {
        long double r, t;

        if (!finitel(x))
                return (x + x);         /* sinh of NaN or +-INF is itself */

#endif /* ! codereview */
        r = fabsl(x);

#endif /* ! codereview */
        if (r < lnovftL) {
                t = expm1l(r);
                r = copysignl((t + t / (one + t)) * half, x);
        } else {
                r = copysignl(expl((r - MEP1 * ln2hi) - MEP1 * ln2lo), x);
                r = scalbnl(r, ME);
        }

#endif /* ! codereview */
        return (r);
}
```

```
**********************************************************
    2675 Thu Jun  6 20:38:58 2019
new/usr/src/lib/libm/common/Q/sinl.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

/*
 * sinl(x)
 * Table look-up algorithm by K.C. Ng, November, 1989.
 *
 * kernel function:
 *      __k_sinl                ... sin function on [-pi/4,pi/4]
 *      __k_cosl                ... cos function on [-pi/4,pi/4]
 *      __rem_pio2l     ... argument reduction routine
 *
 * Method.
 *      Let S and C denote the sin and cos respectively on [-PI/4, +PI/4].
 *      1. Assume the argument x is reduced to y1+y2 = x-k*pi/2 in
 *         [-pi/2 , +pi/2], and let n = k mod 4.
 *      2. Let S=S(y1+y2), C=C(y1+y2). Depending on n, we have
 *
 *          n        sin(x)      cos(x)        tan(x)
 *     ----------------------------------------------------------
 *          0          S           C             S/C
 *          1          C          -S            -C/S
 *          2         -S          -C             S/C
 *          3         -C           S            -C/S
 *     ----------------------------------------------------------
 *
 * Special cases:
 *      Let trig be any of sin, cos, or tan.
 *      trig(+-INF)  is NaN, with signals;
 *      trig(NaN)    is that NaN;
 *
 * Accuracy:
 *      computer TRIG(x) returns trig(x) nearly rounded.
```



```
 */

#pragma weak __sinl = sinl

#include "libm.h"
#include "longdouble.h"

long double
sinl(long double x)
{
sinl(long double x) {
        long double y[2], z = 0.0L;
        int n, ix;

        ix = *(int *)&x;                /* High word of x */
        ix = *(int *) &x;               /* High word of x */
        ix &= 0x7fffffff;

        if (ix <= 0x3ffe9220) {         /* |x| ~< pi/4 */
        if (ix <= 0x3ffe9220)           /* |x| ~< pi/4 */
                return (__k_sinl(x, z));
        } else if (ix >= 0x7fff0000) {  /* sin(Inf or NaN) is NaN */
        else if (ix >= 0x7fff0000)      /* sin(Inf or NaN) is NaN */
                return (x - x);
        } else {                        /* argument reduction needed */
        else {                          /* argument reduction needed */
                n = __rem_pio2l(x, y);

#endif /* ! codereview */
                switch (n & 3) {
                case 0:
                        return (__k_sinl(y[0], y[1]));
                case 1:
                        return (__k_cosl(y[0], y[1]));
                case 2:
                        return (-__k_sinl(y[0], y[1]));
                case 3:
                        return (-__k_cosl(y[0], y[1]));
                }
        }

#endif /* ! codereview */
        /* NOTREACHED */
        return (0.0L);
    return 0.0L;
}
_______unchanged_portion_omitted_
```

```
**********************************************************
    5593 Thu Jun  6 20:38:59 2019
new/usr/src/lib/libm/common/Q/sinpil.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

/*
 * long double sinpil(long double x),
 * return long double precision sinl(pi*x).
 *
 * Algorithm, 10/17/2002, K.C. Ng
 * ------------------------------
 * Let y = |4x|, z = floor(y), and n = (int)(z mod 8.0) (displayed in binary).
 *      1. If y == z, then x is a multiple of pi/4. Return the following values:
 *              ----------------------------------------------------
 *                n  x mod 2    sin(x*pi)    cos(x*pi)   tan(x*pi)
 *              ----------------------------------------------------
 *               000  0.00       +0 ___       +1 ___       +0
 *               001  0.25       +\/0.5       +\/0.5       +1
 *               010  0.50       +1 ___       +0 ___       +inf
 *               011  0.75       +\/0.5       -\/0.5       -1
 *               100  1.00       -0 ___       -1 ___       +0
 *               101  1.25       -\/0.5       -\/0.5       +1
 *               110  1.50       -1 ___       -0 ___       +inf
 *               111  1.75       -\/0.5       +\/0.5       -1
 *              ----------------------------------------------------
 *      2. Otherwise,
 *              ----------------------------------------------------
 *                n     t        sin(x*pi)    cos(x*pi)   tan(x*pi)
 *              ----------------------------------------------------
 *               000  (y-z)/4     sinpi(t)     cospi(t)    tanpi(t)
 *               001  (z+1-y)/4   cospi(t)     sinpi(t)    1/tanpi(t)
 *               010  (y-z)/4     cospi(t)    -sinpi(t)   -1/tanpi(t)
 *               011  (z+1-y)/4   sinpi(t)    -cospi(t)   -tanpi(t)
 *               100  (y-z)/4    -sinpi(t)    -cospi(t)    tanpi(t)
 *               101  (z+1-y)/4  -cospi(t)    -sinpi(t)    1/tanpi(t)
```



```
 *               110  (y-z)/4    -cospi(t)     sinpi(t)   -1/tanpi(t)
 *               111  (z+1-y)/4  -sinpi(t)     cospi(t)   -tanpi(t)
 *              ----------------------------------------------------
 *
 * NOTE. This program compute sinpi/cospi(t<0.25) by __k_sin/cos(pi*t, 0.0).
 * This will return a result with error slightly more than one ulp (but less
 * than 2 ulp). If one wants accurate result,  one may break up pi*t in
 * high (tpi_h) and low (tpi_l) parts and call __k_sin/cos(tip_h, tip_lo)
 * instead.
 */

#include "libm.h"
#include "longdouble.h"

#define I(q, m)                         ((int *)&(q))[m]
#define U(q, m)                         ((unsigned *)&(q))[m]
#define I(q, m) ((int *) &(q))[m]
#define U(q, m) ((unsigned *) &(q))[m]
#if defined(__LITTLE_ENDIAN) || defined(__x86)
#define LDBL_MOST_SIGNIF_I(ld)          ((I(ld, 2) << 16) | (0xffff & (I(ld, \
        1) >> 15)))
#define LDBL_MOST_SIGNIF_I(ld)  ((I(ld, 2) << 16) | (0xffff & (I(ld, 1) >> 15)))
#define LDBL_LEAST_SIGNIF_U(ld)         U(ld, 0)
#define PREC                            64
#define PRECM1                          63
#define PRECM2                          62

#endif /* ! codereview */
static const long double twoPRECM2 = 9.22337203685477580800000000000000e+18L;
#else
#define LDBL_MOST_SIGNIF_I(ld)          I(ld, 0)
#define LDBL_LEAST_SIGNIF_U(ld)         U(ld, sizeof (long double) / \
        sizeof (int) - 1)
#define LDBL_LEAST_SIGNIF_U(ld) U(ld, sizeof (long double) / sizeof (int) - 1)
#define PREC                            113
#define PRECM1                          112
#define PRECM2                          111

#endif /* ! codereview */
static const long double twoPRECM2 = 5.19229685853482762853049632922009 6e+33L;
#endif

static const long double zero = 0.0L,
        quater = 0.25L,
        one = 1.0L,
        pi = 3.14159265358979323846264338327950288419 7e+0000L,
        sqrth = 0.70710678118654752440084436210484903928483593768847 4,
        tiny = 1.0e-100;
static const long double
zero    = 0.0L,
quater  = 0.25L,
one     = 1.0L,
pi      = 3.14159265358979323846264338327950288419 7e+0000L,
sqrth   = 0.70710678118654752440084436210484903928483593768847 4,
tiny    = 1.0e-100;

long double
sinpil(long double x)
{
sinpil(long double x) {
        long double y, z, t;
        int hx, n, k;
        unsigned lx;

        hx = LDBL_MOST_SIGNIF_I(x);
        lx = LDBL_LEAST_SIGNIF_U(x);
```

```
        k = ((hx & 0x7fff0000) >> 16) - 0x3fff;

#endif /* ! codereview */
        if (k >= PRECM2) {              /* |x| >= 2**(Prec-2) */
                if (k >= 16384) {
                if (k >= 16384)
                        y = x - x;
                } else {
                        if (k >= PREC) {
                if (k >= PREC)
                                y = zero;
                        } else if (k == PRECM1) {
                                y = (lx & 1) == 0 ? zero : -zero;
                        } else { /* k = Prec - 2 */
                        else if (k == PRECM1)
                                y = (lx & 1) == 0 ? zero: -zero;
                        else {  /* k = Prec - 2 */
                                y = (lx & 1) == 0 ? zero : one;

#endif /* ! codereview */
                                if ((lx & 2) != 0)
                                        y = -y;
                        }
                }
        } else if (k < -2) {            /* |x| < 0.25 */
        } else if (k < -2)      /* |x| < 0.25 */
                y = __k_sinl(pi * fabsl(x), zero);
        } else {
        else {
                /* y = |4x|, z = floor(y), and n = (int)(z mod 8.0) */
                y = 4.0L * fabsl(x);

#endif /* ! codereview */
                if (k < PRECM2) {
                        z = y + twoPRECM2;
                        n = LDBL_LEAST_SIGNIF_U(z) & 7; /* 3 LSb of z */
                        t = z - twoPRECM2;
                        k = 0;

                        if (t == y) {
                        if (t == y)
                                k = 1;
                        } else if (t > y) {
                        else if (t > y) {
                                n -= 1;
                                t = quater + (y - t) * quater;
                        } else {
                        } else
                                t = (y - t) * quater;
                        }
#endif /* ! codereview */
                } else {                                /* k = Prec-3 */
                        n = LDBL_LEAST_SIGNIF_U(y) & 7; /* 3 LSb of z */
                        k = 1;
                }

#endif /* ! codereview */
                if (k) {                /* x = N/4 */
                        if ((n & 1) != 0)
                                y = sqrth + tiny;
                        else
                                y = (n & 2) == 0 ? zero : one;

#endif /* ! codereview */
                        if ((n & 4) != 0)
```



```
                                y = -y;
                } else {
                        if ((n & 1) != 0)
                                t = quater - t;

#endif /* ! codereview */
                        if (((n + (n & 1)) & 2) == 0)
                                y = __k_sinl(pi * t, zero);
                        else
                                y = __k_cosl(pi * t, zero);

#endif /* ! codereview */
                        if ((n & 4) != 0)
                                y = -y;
                }
        }

#endif /* ! codereview */
        return (hx >= 0 ? y : -y);
}

#endif /* ! codereview */
#undef U
#undef LDBL_LEAST_SIGNIF_U
#undef I
#undef LDBL_MOST_SIGNIF_I
```

```
**********************************************************
    9967 Thu Jun  6 20:38:59 2019
new/usr/src/lib/libm/common/Q/sqrtl.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __sqrtl = sqrtl

#include "libm.h"
#include "longdouble.h"

extern int __swapTE(int);
extern int __swapEX(int);
extern enum fp_direction_type __swapRD(enum fp_direction_type);

/*
 * in struct longdouble, msw consists of
 *      unsigned short  sgn:1;
 *      unsigned short  exp:15;
 *      unsigned short  frac1:16;
 */

#ifdef __LITTLE_ENDIAN

/* array indices used to access words within a double */
#define HIWORD          1
#define LOWORD          0

/* structure used to access words within a quad */
union longdouble {
        struct {
                unsigned int frac4;
                unsigned int frac3;
                unsigned int frac2;
                unsigned int msw;
        } l;
```



```
        long double d;
};

/* default NaN returned for sqrt(neg) */
static const union longdouble qnan = {
        0xffffffff, 0xffffffff, 0xffffffff, 0x7fffffff
};
static const union longdouble
        qnan = { 0xffffffff, 0xffffffff, 0xffffffff, 0x7fffffff };

/* signalling NaN used to raise invalid */
static const union {
        unsigned u[2];
        double d;
} snan = { 0, 0x7ff00001 };

#else

/* array indices used to access words within a double */
#define HIWORD          0
#define LOWORD          1

/* structure used to access words within a quad */
union longdouble {
        struct {
                unsigned int msw;
                unsigned int frac2;
                unsigned int frac3;
                unsigned int frac4;
        } l;
        long double d;
};

/* default NaN returned for sqrt(neg) */
static const union longdouble qnan = {
        0x7fffffff, 0xffffffff, 0xffffffff, 0xffffffff
};
static const union longdouble
        qnan = { 0x7fffffff, 0xffffffff, 0xffffffff, 0xffffffff };

/* signalling NaN used to raise invalid */
static const union {
        unsigned u[2];
        double d;
} snan = { 0x7ff00001, 0 };

#endif /* __LITTLE_ENDIAN */

static const double zero = 0.0,

static const double
        zero = 0.0,
        half = 0.5,
        one = 1.0,
        huge = 1.0e300,
        tiny = 1.0e-300,
        two36 = 6.87194767360000000000e+10,
        two30 = 1.07374182400000000000e+09,
        two6 = 6.40000000000000000000e+01,
        two4 = 1.60000000000000000000e+01,
        twom18 = 3.81469726562500000000e-06,
        twom28 = 3.72529029846191406250e-09,
        twom42 = 2.27373675443232059479e-13,
        twom60 = 8.67361737988403547206e-19,
        twom62 = 2.16840434497100886801e-19,
        twom66 = 1.35525271560688054251e-20,
```

```
117         twom90 = 8.07793566946316088742e-28,
118         twom113 = 9.62964972193617926528e-35,
119         twom124 = 4.70197740328915003187e-38;

121 /*
122  *      Extract the exponent and normalized significand (represented as
123  *      an array of five doubles) from a finite, nonzero quad.
124  */
103 *       Extract the exponent and normalized significand (represented as
104 *       an array of five doubles) from a finite, nonzero quad.
105 */
125 static int
126 __q_unpack(const union longdouble *x, double *s)
127 {
128         union {
129                 double d;
130                 unsigned int l[2];
131         } u;

133 #endif /* ! codereview */
134         double b;
135         unsigned int lx, w[3];
136         int ex;

138         /* get the normalized significand and exponent */
139         ex = (int)((x->l.msw & 0x7fffffff) >> 16);
113         ex = (int) ((x->l.msw & 0x7fffffff) >> 16);
140         lx = x->l.msw & 0xffff;

142         if (ex) {
115         if (ex)
116         {
143                 lx |= 0x10000;
144                 w[0] = x->l.frac2;
145                 w[1] = x->l.frac3;
146                 w[2] = x->l.frac4;
147         } else {
148                 if (lx | (x->l.frac2 & 0xfffe0000)) {
121         }
122         else
123         {
124                 if (lx | (x->l.frac2 & 0xfffe0000))
125                 {
149                         w[0] = x->l.frac2;
150                         w[1] = x->l.frac3;
151                         w[2] = x->l.frac4;
152                         ex = 1;
153                 } else if (x->l.frac2 | (x->l.frac3 & 0xfffe0000)) {
130                 }
131                 else if (x->l.frac2 | (x->l.frac3 & 0xfffe0000))
132                 {
154                         lx = x->l.frac2;
155                         w[0] = x->l.frac3;
156                         w[1] = x->l.frac4;
157                         w[2] = 0;
158                         ex = -31;
159                 } else if (x->l.frac3 | (x->l.frac4 & 0xfffe0000)) {
138                 }
139                 else if (x->l.frac3 | (x->l.frac4 & 0xfffe0000))
140                 {
160                         lx = x->l.frac3;
161                         w[0] = x->l.frac4;
162                         w[1] = w[2] = 0;
163                         ex = -63;
164                 } else {
```



```
145                 }
146                 else
147                 {
165                         lx = x->l.frac4;
166                         w[0] = w[1] = w[2] = 0;
167                         ex = -95;
168                 }

170                 while ((lx & 0x10000) == 0) {
152                 while ((lx & 0x10000) == 0)
153                 {
171                         lx = (lx << 1) | (w[0] >> 31);
172                         w[0] = (w[0] << 1) | (w[1] >> 31);
173                         w[1] = (w[1] << 1) | (w[2] >> 31);
174                         w[2] <<= 1;
175                         ex--;
176                 }
177         }

179         /* extract the significand into five doubles */
180         u.l[HIWORD] = 0x42300000;
181         u.l[LOWORD] = 0;
182         b = u.d;
183         u.l[LOWORD] = lx;
184         s[0] = u.d - b;

186         u.l[HIWORD] = 0x40300000;
187         u.l[LOWORD] = 0;
188         b = u.d;
189         u.l[LOWORD] = w[0] & 0xffffff00;
190         s[1] = u.d - b;

192         u.l[HIWORD] = 0x3e300000;
193         u.l[LOWORD] = 0;
194         b = u.d;
195         u.l[HIWORD] |= w[0] & 0xff;
196         u.l[LOWORD] = w[1] & 0xffff0000;
197         s[2] = u.d - b;

199         u.l[HIWORD] = 0x3c300000;
200         u.l[LOWORD] = 0;
201         b = u.d;
202         u.l[HIWORD] |= w[1] & 0xffff;
203         u.l[LOWORD] = w[2] & 0xff000000;
204         s[3] = u.d - b;

206         u.l[HIWORD] = 0x3c300000;
207         u.l[LOWORD] = 0;
208         b = u.d;
209         u.l[LOWORD] = w[2] & 0xffffff;
210         s[4] = u.d - b;

212         return (ex - 0x3fff);
195         return ex - 0x3fff;
213 }

215 /*
216  *      Pack an exponent and array of three doubles representing a finite,
217  *      nonzero number into a quad.  Assume the sign is already there and
218  *      the rounding mode has been fudged accordingly.
219  */
200 *       Pack an exponent and array of three doubles representing a finite,
201 *       nonzero number into a quad.  Assume the sign is already there and
202 *       the rounding mode has been fudged accordingly.
203 */
```

```
220 static void
221 __q_pack(const double *z, int exp, enum fp_direction_type rm,
222     union longdouble *x, int *inexact)
223 {
224         union {
225                 double d;
226                 unsigned int l[2];
227         } u;

229 #endif /* ! codereview */
230         double s[3], t, t2;
231         unsigned int msw, frac2, frac3, frac4;

233         /* bias exponent and strip off integer bit */
234         exp += 0x3fff;
235         s[0] = z[0] - one;
236         s[1] = z[1];
237         s[2] = z[2];

239         /*
240          * chop the significand to obtain the fraction;
241          * use round-to-minus-infinity to ensure chopping
242          */
243         (void) __swapRD(fp_negative);

245         /* extract the first eighty bits of fraction */
246         t = s[1] + s[2];
247         u.d = two36 + (s[0] + t);
248         msw = u.l[LOWORD];
249         s[0] -= (u.d - two36);

251         u.d = two4 + (s[0] + t);
252         frac2 = u.l[LOWORD];
253         s[0] -= (u.d - two4);

255         u.d = twom28 + (s[0] + t);
256         frac3 = u.l[LOWORD];
257         s[0] -= (u.d - twom28);

259         /* condense the remaining fraction; errors here won't matter */
260         t = s[0] + s[1];
261         s[1] = ((s[0] - t) + s[1]) + s[2];
262         s[0] = t;

264         /* get the last word of fraction */
265         u.d = twom60 + (s[0] + s[1]);
266         frac4 = u.l[LOWORD];
267         s[0] -= (u.d - twom60);

269         /*
270          * keep track of what's left for rounding; note that
271          * t2 will be non-negative due to rounding mode
272          */
273         t = s[0] + s[1];
274         t2 = (s[0] - t) + s[1];

276         if (t != zero) {
212         if (t != zero)
213         {
277                 *inexact = 1;

279                 /* decide whether to round the fraction up */
280                 if (rm == fp_positive || (rm == fp_nearest && (t > twom113 ||
281                     (t == twom113 && (t2 != zero || frac4 & 1))))) {
218                         (t == twom113 && (t2 != zero || frac4 & 1)))))
219                 {
```



```
282                         /* round up and renormalize if necessary */
283                         if (++frac4 == 0) {
284                                 if (++frac3 == 0) {
285                                         if (++frac2 == 0) {
286                                                 if (++msw == 0x10000) {
221                         if (++frac4 == 0)
222                                 if (++frac3 == 0)
223                                         if (++frac2 == 0)
224                                                 if (++msw == 0x10000)
225                                                 {
287                                                         msw = 0;
288                                                         exp++;
289                                                 }
290                                         }
291                                 }
292                         }
293                 }
294         }
295 #endif /* ! codereview */

297         /* assemble the result */
298         x->l.msw |= msw | (exp << 16);
299         x->l.frac2 = frac2;
300         x->l.frac3 = frac3;
301         x->l.frac4 = frac4;
302 }


304 /*
305  *      Compute the square root of x and place the TP result in s.
306  */
233 *       Compute the square root of x and place the TP result in s.
234 */
307 static void
308 __q_tp_sqrt(const double *x, double *s)
309 {
310         double c, rr, r[3], tt[3], t[5];

312         /* approximate the divisor for the Newton iteration */
313         c = sqrt((x[0] + x[1]) + x[2]);
314         rr = half / c;

316         /* compute the first five "digits" of the square root */
317         t[0] = (c + two30) - two30;
318         tt[0] = t[0] + t[0];
319         r[0] = ((x[0] - t[0] * t[0]) + x[1]) + x[2];

321         t[1] = (rr * (r[0] + x[3]) + two6) - two6;
322         tt[1] = t[1] + t[1];
323         r[0] -= tt[0] * t[1];
324         r[1] = x[3] - t[1] * t[1];
325         c = (r[1] + twom18) - twom18;
326         r[0] += c;
327         r[1] = (r[1] - c) + x[4];

329         t[2] = (rr * (r[0] + r[1]) + twom18) - twom18;
330         tt[2] = t[2] + t[2];
331         r[0] -= tt[0] * t[2];
332         r[1] -= tt[1] * t[2];
333         c = (r[1] + twom42) - twom42;
334         r[0] += c;
335         r[1] = (r[1] - c) - t[2] * t[2];

337         t[3] = (rr * (r[0] + r[1]) + twom42) - twom42;
338         r[0] = ((r[0] - tt[0] * t[3]) + r[1]) - tt[1] * t[3];
339         r[1] = -tt[2] * t[3];
```

```
340         c = (r[1] + twom90) - twom90;
341         r[0] += c;
342         r[1] = (r[1] - c) - t[3] * t[3];

344         t[4] = (rr * (r[0] + r[1]) + twom66) - twom66;

346         /* here we just need to get the sign of the remainder */
347         c = (((((r[0] - tt[0] * t[4]) - tt[1] * t[4]) + r[1]) - tt[2] * t[4]) -
348             (t[3] + t[3]) * t[4]) - t[4] * t[4];
275         c = (((((r[0] - tt[0] * t[4]) - tt[1] * t[4]) + r[1])
276                 - tt[2] * t[4]) - (t[3] + t[3]) * t[4]) - t[4] * t[4];

350         /* reduce to three doubles */
351         t[0] += t[1];
352         t[1] = t[2] + t[3];
353         t[2] = t[4];

355         /* if the third term might lie on a rounding boundary, perturb it */
356         if (c != zero && t[2] == (twom62 + t[2]) - twom62) {
284         if (c != zero && t[2] == (twom62 + t[2]) - twom62)
285         {
357                 if (c < zero)
358                         t[2] -= twom124;
359                 else
360                         t[2] += twom124;
361         }

363         /* condense the square root */
364         c = t[1] + t[2];
365         t[2] += (t[1] - c);
366         t[1] = c;
367         c = t[0] + t[1];
368         s[1] = t[1] + (t[0] - c);
369         s[0] = c;

371         if (s[1] == zero) {
299         if (s[1] == zero)
300         {
372                 c = s[0] + t[2];
373                 s[1] = t[2] + (s[0] - c);
374                 s[0] = c;
375                 s[2] = zero;
376         } else {
305         }
306         else
307         {
377                 c = s[1] + t[2];
378                 s[2] = t[2] + (s[1] - c);
379                 s[1] = c;
380         }
381 }

383 long double
384 sqrtl(long double ldx)
385 {
386         union   longdouble x;
387         volatile double t;
388         double xx[5], zz[3];
389         enum fp_direction_type rm;
390         int ex, inexact, exc, traps;

392         /* clear cexc */
393         t = zero;
394         t -= zero;
```



```
396         /* check for zero operand */
397         x.d = ldx;

399 #endif /* ! codereview */
400         if (!((x.l.msw & 0x7fffffff) | x.l.frac2 | x.l.frac3 | x.l.frac4))
401                 return (ldx);
330                 return ldx;

403         /* handle nan and inf cases */
404         if ((x.l.msw & 0x7fffffff) >= 0x7fff0000) {
405                 if ((x.l.msw & 0xffff) | x.l.frac2 | x.l.frac3 | x.l.frac4) {
406                         if (!(x.l.msw & 0x8000)) {
333         if ((x.l.msw & 0x7fffffff) >= 0x7fff0000)
334         {
335                 if ((x.l.msw & 0xffff) | x.l.frac2 | x.l.frac3 | x.l.frac4)
336                 {
337                         if (!(x.l.msw & 0x8000))
338                         {
407                                 /* snan, signal invalid */
408                                 t += snan.d;
409                         }

411 #endif /* ! codereview */
412                         x.l.msw |= 0x8000;
413                         return (x.d);
342                         return x.d;
414                 }

416                 if (x.l.msw & 0x80000000) {
344                 if (x.l.msw & 0x80000000)
345                 {
417                         /* sqrt(-inf), signal invalid */
418                         t = -one;
419                         t = sqrt(t);
420                         return (qnan.d);
349                         return qnan.d;
421                 }

423 #endif /* ! codereview */
424                 /* sqrt(inf), return inf */
425                 return (x.d);
351                 return x.d;
426         }

428         /* handle negative numbers */
429         if (x.l.msw & 0x80000000) {
355         if (x.l.msw & 0x80000000)
356         {
430                 t = -one;
431                 t = sqrt(t);
432                 return (qnan.d);
359                 return qnan.d;
433         }

435         /* now x is finite, positive */

437         traps = __swapTE(0);
438         exc = __swapEX(0);
439         rm = __swapRD(fp_nearest);

441         ex = __q_unpack(&x, xx);

443         if (ex & 1) {
369         if (ex & 1)
370         {
444                 /* make exponent even */
```

```
                xx[0] += xx[0];
                xx[1] += xx[1];
                xx[2] += xx[2];
                xx[3] += xx[3];
                xx[4] += xx[4];
                ex--;
        }

#endif /* ! codereview */
        __q_tp_sqrt(xx, zz);

        /* put everything together */
        x.l.msw = 0;
        inexact = 0;
        __q_pack(zz, ex >> 1, rm, &x, &inexact);

        (void) __swapRD(rm);
        (void) __swapEX(exc);
        (void) __swapTE(traps);

        if (inexact) {
        if (inexact)
        {
                t = huge;
                t += tiny;
        }

        return (x.d);
        return x.d;
}
```

_____*unchanged_portion_omitted_*

```
**********************************************************
    2635 Thu Jun  6 20:39:00 2019
new/usr/src/lib/libm/common/Q/tanhl.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __tanhl = tanhl

/*
 * tanhl(x) returns the hyperbolic tangent of x
 *
 * Method :
 *      1. reduce x to non-negative:  tanhl(-x) = - tanhl(x).
 *      2.
 *        0      <  x <=  small    :  tanhl(x) := x
 *                                                 -expm1l(-2x)
 *        small  <  x <=  1        :  tanhl(x) := --------------
 *                                                expm1l(-2x) + 2
 *                                                       2
 *        1      <= x <= threshold :  tanhl(x) := 1 -  ---------------
 *                                                   expm1l(2x) + 2
 *     threshold <  x <= INF       :  tanhl(x) := 1.
 *
 * where
 *      single :        small = 1.e-5           threshold = 11.0
 *      double :        small = 1.e-10          threshold = 22.0
 *      quad   :        small = 1.e-20          threshold = 45.0
 *
 * Note: threshold was chosen so that
 *              fl(1.0+2/(expm1(2*threshold)+2)) == 1.
 *
 * Special cases:
 *      tanhl(NaN) is NaN;
 *      only tanhl(0.0)=0.0 is exact for finite argument.
 */
```



```
#include "libm.h"
#include "longdouble.h"

static const long double small = 1.0e-20L,
        one = 1.0,
        two = 2.0,
static const long double small = 1.0e-20L, one = 1.0, two = 2.0,
#ifndef lint
        big = 1.0e+20L,
#endif
        threshold = 45.0L;

long double
tanhl(long double x)
{
tanhl(long double x) {
        long double t, y, z;
        int signx;

#endif /* ! codereview */
#ifndef lint
        volatile long double dummy;
#endif

        if (isnanl(x))
                return (x + x);         /* x is NaN */

#endif /* ! codereview */
        signx = signbitl(x);
        t = fabsl(x);
        z = one;

#endif /* ! codereview */
        if (t <= threshold) {
                if (t > one) {
                if (t > one)
                        z = one - two / (expm1l(t + t) + two);
                } else if (t > small) {
                else if (t > small) {
                        y = expm1l(-t - t);
                        z = -y / (y + two);
                } else {
#ifndef lint
                        dummy = t + big;
                        /* inexact if t != 0 */
#endif
                        return (x);
                }
        } else if (!finitel(t)) {
        } else if (!finitel(t))
                return (copysignl(one, x));
        } else {
        else
                return (signx ? -z + small * small : z - small * small);
        }

#endif /* ! codereview */
        return (signx ? -z : z);
}
```

```
**********************************************************
    2441 Thu Jun  6 20:39:00 2019
new/usr/src/lib/libm/common/Q/tanl.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

/*
 * tanl(x)
 * Table look-up algorithm by K.C. Ng, November, 1989.
 *
 * kernel function:
 *      __k_tanl        ... tangent function on [-pi/4,pi/4]
 *      __rem_pio2l     ... argument reduction routine
 *
 * Method.
 *      Let S and C denote the sin and cos respectively on [-PI/4, +PI/4].
 *      1. Assume the argument x is reduced to y1+y2 = x-k*pi/2 in
 *         [-pi/2 , +pi/2], and let n = k mod 4.
 *      2. Let S=S(y1+y2), C=C(y1+y2). Depending on n, we have
 *
 *          n        sin(x)      cos(x)        tan(x)
 *     ----------------------------------------------------------
 *          0          S           C             S/C
 *          1          C          -S            -C/S
 *          2         -S          -C             S/C
 *          3         -C           S            -C/S
 *     ----------------------------------------------------------
 *
 * Special cases:
 *      Let trig be any of sin, cos, or tan.
 *      trig(+-INF)  is NaN, with signals;
 *      trig(NaN)    is that NaN;
 *
 * Accuracy:
 *      computer TRIG(x) returns trig(x) nearly rounded.
 */
```



```
#pragma weak __tanl = tanl

#include "libm.h"
#include "longdouble.h"

long double
tanl(long double x)
{
tanl(long double x) {
        long double y[2], z = 0.0L;
        int n, ix;

        ix = *(int *)&x;                /* High word of x */
        ix = *(int *) &x;               /* High word of x */
        ix &= 0x7fffffff;

        if (ix <= 0x3ffe9220) {         /* |x| ~< pi/4 */
        if (ix <= 0x3ffe9220)           /* |x| ~< pi/4 */
                return (__k_tanl(x, z, 0));
        } else if (ix >= 0x7fff0000) {  /* trig(Inf or NaN) is NaN */
        else if (ix >= 0x7fff0000)      /* trig(Inf or NaN) is NaN */
                return (x - x);
        } else {                        /* argument reduction needed */
        else {                          /* argument reduction needed */
                n = __rem_pio2l(x, y);
                return (__k_tanl(y[0], y[1], (n & 1)));
        }
}
______unchanged_portion_omitted_
```

```
**********************************************************
    3473 Thu Jun  6 20:39:01 2019
new/usr/src/lib/libm/common/R/_TBL_r_atan_.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

/*
 * Table of constants for r_atan_().
 * By K.C. Ng, March 9, 1989
 */

#include "libm.h"

const float _TBL_r_atan_hi[] = {
        4.636476040e-01, 4.883339405e-01, 5.123894811e-01, 5.358112454e-01,
        5.585992932e-01, 5.807563663e-01, 6.022873521e-01, 6.231993437e-01,
        6.435011029e-01, 6.632030010e-01, 6.823165417e-01, 7.008544207e-01,
        7.188299894e-01, 7.362574339e-01, 7.531512976e-01, 7.695264816e-01,
        7.853981853e-01, 8.156919479e-01, 8.441540003e-01, 8.709034324e-01,
        8.960554004e-01, 9.197195768e-01, 9.420000315e-01, 9.629943371e-01,
        9.827937484e-01, 1.001483083e+00, 1.019141316e+00, 1.035841227e+00,
        1.051650167e+00, 1.066630363e+00, 1.080839038e+00, 1.094328880e+00,
        1.107148767e+00, 1.130953789e+00, 1.152572036e+00, 1.172273874e+00,
        1.190289974e+00, 1.206817389e+00, 1.222025275e+00, 1.236059427e+00,
        1.249045730e+00, 1.261093378e+00, 1.272297382e+00, 1.282740831e+00,
        1.292496681e+00, 1.301628828e+00, 1.310193896e+00, 1.318242073e+00,
        1.325817704e+00, 1.339705706e+00, 1.352127433e+00, 1.363300085e+00,
        1.373400807e+00, 1.382574797e+00, 1.390942812e+00, 1.398605466e+00,
        1.405647635e+00, 1.412141085e+00, 1.418146968e+00, 1.423717976e+00,
        1.428899288e+00, 1.433730125e+00, 1.438244820e+00, 1.442473054e+00,
        1.446441293e+00,
};

const float _TBL_r_atan_lo[] = {
        +5.012158688e-09, +1.055042365e-08, -2.075691974e-08, -7.480973174e-09,
        +2.211159789e-08, -1.268522887e-08, -5.950149262e-09, -1.374726910e-08,
```



```
        +5.868937336e-09, -8.316245470e-09, +1.320299514e-08, -1.277747597e-08,
        +1.018833551e-08, -4.909868068e-09, -1.660708016e-08, -1.222759671e-09,
        -2.185569414e-08, -2.462078896e-08, -1.416911655e-08, +2.470642002e-08,
        -1.580020736e-08, +2.851478520e-08, +8.908211058e-09, -6.400973085e-09,
        -2.513142405e-08, +5.292293181e-08, +2.785247055e-08, +2.643104224e-08,
        +4.603683834e-08, +1.851388043e-09, -3.735403453e-08, +2.701113111e-08,
        -4.872354964e-08, -4.477816518e-08, -3.857382325e-08, +6.845639611e-09,
        -2.453011483e-08, -1.824929363e-08, +4.798058129e-08, +6.221672777e-08,
        +4.276110843e-08, +4.185424007e-09, +1.285398099e-08, +4.836914869e-08,
        -1.342359379e-08, +5.960489879e-09, +3.875391386e-08, -2.204224536e-08,
        -4.053271141e-08, -4.604370218e-08, -5.190222652e-08, +1.529194549e-08,
        -4.043566193e-08, +2.481348993e-08, +1.503647518e-08, +4.638297924e-08,
        +1.392036975e-08, -2.006252586e-08, +3.051175312e-08, -4.209960824e-09,
        -1.598675681e-08, +2.705746205e-08, -2.514289044e-08, +4.517691110e-08,
        +3.948537852e-08,
};
```

```
**********************************************************
    2757 Thu Jun  6 20:39:01 2019
new/usr/src/lib/libm/common/R/__cosf.c
11210 libm should be cstyle(10NBLD) clean
**********************************************************
   1 /*
   2  * CDDL HEADER START
   3  *
   4  * The contents of this file are subject to the terms of the
   5  * Common Development and Distribution License (the "License").
   6  * You may not use this file except in compliance with the License.
   7  *
   8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
   9  * or http://www.opensolaris.org/os/licensing.
  10  * See the License for the specific language governing permissions
  11  * and limitations under the License.
  12  *
  13  * When distributing Covered Code, include this CDDL HEADER in each
  14  * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  15  * If applicable, add the following below this CDDL HEADER, with the
  16  * fields enclosed by brackets "[]" replaced with your own identifying
  17  * information: Portions Copyright [yyyy] [name of copyright owner]
  18  *
  19  * CDDL HEADER END
  20  */

  22 /*
  23  * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
  24  */

  26 #endif /* ! codereview */
  27 /*
  28  * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
  29  * Use is subject to license terms.
  30  */

  32 #include "libm.h"


  25 /* INDENT OFF */
  35 /*
  36  * float __k_cos(double x);
  37  * kernel (float) cos function on [-pi/4, pi/4], pi/4 ~ 0.785398164
  38  * Input x is in double and assumed to be bounded by ~pi/4 in magnitude.
  39  *
  40  * Method: Let z = x * x, then
  41  *      C(x) = (C0 + C1*z + C2*z*z) * (C3 + C4*z + z*z)
  42  * where
  43  *      C0 =   1.09349482127188401868272000389539985058873853699e-0003
  44  *      C1 =  -5.03324285989964979398034700054920226866107675091e-0004
  45  *      C2 =   2.43792880266971107750418061559602239831538067410e-0005
  46  *      C3 =   9.14499072605666582228127405245558035523741471271e+0002
  47  *      C4 =  -3.63151270591815439197122504991683846785293207730e+0001
  48  *
  49  * The remez error is bound by  |cos(x) - C(x)| < 2**(-34.2)
  50  *
  51  * Constants:
  52  * The hexadecimal values are the intended ones for the following constants.
  53  * The decimal values may be used, provided that the compiler will convert
  54  * from decimal to binary accurately enough to produce the hexadecimal values
  55  * shown.
  56  */
  48 /* INDENT ON */

  58 static const double q[] = {
  59 /* C0 = */
```



```
  60         1.09349482127188401868272000389539985058873853699e-0003,
  51 /* C0 = */   1.09349482127188401868272000389539985058873853699e-0003,
  61 /* C1 = */ -5.03324285989964979398034700054920226866107675091e-0004,
  62 /* C2 = */ 2.43792880266971107750418061559602239831538067410e-0005,
  63 /* C3 = */ 9.14499072605666582228127405245558035523741471271e+0002,
  64 /* C4 = */ -3.63151270591815439197122504991683846785293207730e+0001, };
  55 /* C4 = */  -3.63151270591815439197122504991683846785293207730e+0001,
  56 };

  66 #define C0              q[0]
  67 #define C1              q[1]
  68 #define C2              q[2]
  69 #define C3              q[3]
  70 #define C4              q[4]

  72 float
  73 __k_cosf(double x)
  74 {
  65 __k_cosf(double x) {
  75         float ft;
  76         double z;
  77         int hx;

  79         hx = ((int *)&x)[HIWORD];               /* hx = leading x  */

  70         hx = ((int *) &x)[HIWORD];      /* hx = leading x  */
  81         if ((hx & ~0x80000000) < 0x3f100000) {  /* |x| < 2**-14 */
  82                 ft = (float)1;

  84                 if (((int)x) == 0)              /* raise inexact if x != 0 */
  72                 ft = (float) 1;
  73                 if (((int) x) == 0)     /* raise inexact if x != 0 */
  85                         return (ft);
  86         }

  88 #endif /* ! codereview */
  89         z = x * x;
  90         ft = (float)(((C0 + z * C1) + (z * z) * C2) * (C3 + z * (C4 + z)));
  76         ft = (float) (((C0 + z * C1) + (z * z) * C2) * (C3 + z * (C4 + z)));
  91         return (ft);
  92 }
______unchanged_portion_omitted_
```

```
**********************************************************
    3597 Thu Jun  6 20:39:02 2019
new/usr/src/lib/libm/common/R/__sincosf.c
11210 libm should be cstyle(10NBLD) clean
**********************************************************
   1 /*
   2  * CDDL HEADER START
   3  *
   4  * The contents of this file are subject to the terms of the
   5  * Common Development and Distribution License (the "License").
   6  * You may not use this file except in compliance with the License.
   7  *
   8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
   9  * or http://www.opensolaris.org/os/licensing.
  10  * See the License for the specific language governing permissions
  11  * and limitations under the License.
  12  *
  13  * When distributing Covered Code, include this CDDL HEADER in each
  14  * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  15  * If applicable, add the following below this CDDL HEADER, with the
  16  * fields enclosed by brackets "[]" replaced with your own identifying
  17  * information: Portions Copyright [yyyy] [name of copyright owner]
  18  *
  19  * CDDL HEADER END
  20  */

  22 /*
  23  * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
  24  */

  26 #endif /* ! codereview */
  27 /*
  28  * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
  29  * Use is subject to license terms.
  30  */

  32 #include "libm.h"


  25 /* INDENT OFF */
  35 /*
  36  * void __k_sincosf(double x, float *s, float *c);
  37  * kernel (float) sincos function on [-pi/4, pi/4], pi/4 ~ 0.785398164
  38  * Input x is in double and assumed to be bounded by ~pi/4 in magnitude.
  39  *
  40  * Method: Let z = x * x, then
  41  *      S(x) = x(S0 + S1*z)(S2 + S3*z + z*z)
  42  *      C(x) = (C0 + C1*z + C2*z*z) * (C3 + C4*z + z*z)
  43  * where
  44  *      S0 =   1.85735322054308378716204874632872525989806770558e-0003
  45  *      S1 =  -1.95035094218403635082921458859320791358115801259e-0004
  46  *      S2 =   5.38400550766074785970952495168558701485841707252e+0002
  47  *      S3 =  -3.31975110777873728964197739157371509422022905947e+0001
  48  *      C0 =   1.09349482127188401868272000389539985058873853699e-0003
  49  *      C1 =  -5.03324285989964979398034700054920226866107675091e-0004
  50  *      C2 =   2.43792880266971107750418061559602239831538067410e-0005
  51  *      C3 =   9.14499072605666582228127405245558035523741471271e+0002
  52  *      C4 =  -3.63151270591815439197122504991683846785293207730e+0001
  53  *
  54  * The remez error in S is bound by  |(sin(x) - S(x))/x| < 2**(-28.2)
  55  * The remez error in C is bound by  |cos(x) - C(x)| < 2**(-34.2)
  56  *
  57  * Constants:
  58  * The hexadecimal values are the intended ones for the following constants.
  59  * The decimal values may be used, provided that the compiler will convert
  60  * from decimal to binary accurately enough to produce the hexadecimal values
```



```
  61  * shown.
  62  */
  54 /* INDENT ON */

  64 static const double q[] = {
  65 /* S0 = */
  66         1.85735322054308378716204874632872525989806770558e-0003,
  57 /* S0 = */  1.85735322054308378716204874632872525989806770558e-0003,
  67 /* S1 = */ -1.95035094218403635082921458859320791358115801259e-0004,
  68 /* S2 = */ 5.38400550766074785970952495168558701485841707252e+0002,
  69 /* S3 = */ -3.31975110777873728964197739157371509422022905947e+0001,
  70 /* C0 = */ 1.09349482127188401868272000389539985058873853699e-0003,
  71 /* C1 = */ -5.03324285989964979398034700054920226866107675091e-0004,
  72 /* C2 = */ 2.43792880266971107750418061559602239831538067410e-0005,
  73 /* C3 = */ 9.14499072605666582228127405245558035523741471271e+0002,
  74 /* C4 = */ -3.63151270591815439197122504991683846785293207730e+0001, };
  65 /* C4 = */ -3.63151270591815439197122504991683846785293207730e+0001,
  66 };


  76 #define S0              q[0]
  77 #define S1              q[1]
  78 #define S2              q[2]
  79 #define S3              q[3]
  80 #define C0              q[4]
  81 #define C1              q[5]
  82 #define C2              q[6]
  83 #define C3              q[7]
  84 #define C4              q[8]

  86 void
  87 __k_sincosf(double x, float *s, float *c)
  88 {
  80 __k_sincosf(double x, float *s, float *c) {
  89         double z;
  90         int hx;

  92         hx = ((int *)&x)[HIWORD];               /* hx = leading x */

  84         hx = ((int *) &x)[HIWORD];      /* hx = leading x */
  94         /* small argument */
  95         if ((hx & ~0x80000000) < 0x3f100000) {  /* if |x| < 2**-14 */
  96                 *s = (float)x;
  97                 *c = (float)1;

  99                 if ((int)x == 0)        /* raise inexact if x != 0 */
  87                 *s = (float) x; *c = (float) 1;
  88                 if ((int) x == 0)       /* raise inexact if x != 0 */
 100                         return;
 101         }

 103 #endif /* ! codereview */
 104         z = x * x;
 105         *s = (float)((x * (S0 + z * S1)) * (S2 + z * (S3 + z)));
 106         *c = (float)(((C0 + z * C1) + (z * z) * C2) * (C3 + z * (C4 + z)));
  91         *s = (float) ((x * (S0 + z * S1)) * (S2 + z * (S3 + z)));
  92         *c = (float) (((C0 + z * C1) + (z * z) * C2) * (C3 + z * (C4 + z)));
 107 }
______unchanged_portion_omitted_
```

```
**********************************************************
    2597 Thu Jun  6 20:39:02 2019
new/usr/src/lib/libm/common/R/__sinf.c
11210 libm should be cstyle(10NBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#include "libm.h"


/* INDENT OFF */
/*
 * float __k_sin(double x);
 * kernel (float) sin function on [-pi/4, pi/4], pi/4 ~ 0.785398164
 * Input x is in double and assumed to be bounded by ~pi/4 in magnitude.
 *
 * Method: Let z = x * x, then
 *      S(x) = x(S0 + S1*z)(S2 + S3*z + z*z)
 * where
 *      S0 =   1.85735322054308378716204874632872525989806770558e-0003,
 *      S1 =  -1.95035094218403635082921458859320791358115801259e-0004,
 *      S2 =   5.38400550766074785970952495168558701485841707252e+0002,
 *      S3 =  -3.31975110777873728964197739157371509422022905947e+0001,
 *
 * The remez error is bound by  |(sin(x) - S(x))/x| < 2**(-28.2)
 *
 * Constants:
 * The hexadecimal values are the intended ones for the following constants.
 * The decimal values may be used, provided that the compiler will convert
 * from decimal to binary accurately enough to produce the hexadecimal values
 * shown.
 */
/* INDENT ON */

static const double q[] = {
/* S0 = */
        1.85735322054308378716204874632872525989806770558e-0003,
```



```
/* S0 = */  1.85735322054308378716204874632872525989806770558e-0003,
/* S1 = */ -1.95035094218403635082921458859320791358115801259e-0004,
/* S2 = */ 5.38400550766074785970952495168558701485841707252e+0002,
/* S3 = */ -3.31975110777873728964197739157371509422022905947e+0001, };
/* S3 = */ -3.31975110777873728964197739157371509422022905947e+0001,
};

#define S0              q[0]
#define S1              q[1]
#define S2              q[2]
#define S3              q[3]

float
__k_sinf(double x)
{
__k_sinf(double x) {
        float ft;
        double z;
        int hx;

        hx = ((int *)&x)[HIWORD];               /* hx = leading x */

        hx = ((int *) &x)[HIWORD];      /* hx = leading x */
        if ((hx & ~0x80000000) < 0x3f100000) {  /* if |x| < 2**-14 */
                ft = (float)x;

                if ((int)x == 0)                /* raise inexact if x != 0 */
                ft = (float) x;
                if ((int) x == 0)       /* raise inexact if x != 0 */
                        return (ft);
        }

#endif /* ! codereview */
        z = x * x;
        ft = (float)((x * (S0 + z * S1)) * (S2 + z * (S3 + z)));
        ft = (float) ((x * (S0 + z * S1)) * (S2 + z * (S3 + z)));
        return (ft);
}
______unchanged_portion_omitted_
```

```
**********************************************************
    2993 Thu Jun  6 20:39:03 2019
new/usr/src/lib/libm/common/R/__tanf.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#include "libm.h"


/* INDENT OFF */
/*
 * float __k_tan(double x);
 * kernel (float) tan function on [-pi/4, pi/4], pi/4 ~ 0.785398164
 * Input x is in double and assumed to be bounded by ~pi/4 in magnitude.
 *
 * Constants:
 * The hexadecimal values are the intended ones for the following constants.
 * The decimal values may be used, provided that the compiler will convert
 * from decimal to binary accurately enough to produce the hexadecimal values
 * shown.
 */

static const double q[] = {
/* one */
        1.0,
/* one */ 1.0,
/* P0 */ 4.460669284289592306791405462718103080987930297 85e-0003,
/* P1 */ 4.921653163091890270663952833274379372596740722 66e+0000,
/* P2 */ -7.114106481614734800444921347661875188350677490 23e-0001,
/* P3 */ 4.085498083740533914465231646317988634109497070 31e+0000,
/* P4 */ 2.504110703980509278210320189828053116798400878 91e+0000,
/* P5 */ 1.114920645602511584115745790768414735794067382 81e+0001,
/* P6 */ -1.505655409684226508915116937714628875255584716 80e+0000,
/* P7 */ -1.814843788783492950500431106775067746639251708 98e+0000,
/* T0 */ 3.333335997532835641297409611782510896641e-0001,
/* T1 */ 2.999997598248363761541668282006867229939e+00,
```



```
};

/* INDENT ON */

#define one             q[0]
#define P0              q[1]
#define P1              q[2]
#define P2              q[3]
#define P3              q[4]
#define P4              q[5]
#define P5              q[6]
#define P6              q[7]
#define P7              q[8]
#define T0              q[9]
#define T1              q[10]

float
__k_tanf(double x, int n)
{
__k_tanf(double x, int n) {
        float ft = 0.0;
        double z, w;
        int ix;

        ix = ((int *)&x)[HIWORD] & ~0x80000000; /* ix = leading |x| */

        ix = ((int *) &x)[HIWORD] & ~0x80000000;        /* ix = leading |x| */
        /* small argument */
        if (ix < 0x3f800000) {                  /* if |x| < 0.0078125 = 2**-7 */
                if (ix < 0x3f100000) {          /* if |x| < 2**-14 */
                        if ((int)x == 0)        /* raise inexact if x != 0 */
                                ft = n == 0 ? (float)x : (float)(-one / x);

                        if ((int) x == 0) {     /* raise inexact if x != 0 */
                                ft = n == 0 ? (float) x : (float) (-one / x);
                        }
                        return (ft);
                }

#endif /* ! codereview */
                z = (x * T0) * (T1 + x * x);
                ft = n == 0 ? (float)z : (float)(-one / z);
                ft = n == 0 ? (float) z : (float) (-one / z);
                return (ft);
        }

#endif /* ! codereview */
        z = x * x;
        w = ((P0 * x) * (P1 + z * (P2 + z)) * (P3 + z * (P4 + z))) * (P5 + z *
            (P6 + z * (P7 + z)));
        ft = n == 0 ? (float)w : (float)(-one / w);
        w = ((P0 * x) * (P1 + z * (P2 + z)) * (P3 + z * (P4 + z)))
                * (P5 + z * (P6 + z * (P7 + z)));
        ft = n == 0 ? (float) w : (float) (-one / w);
        return (ft);
}
_______unchanged_portion_omitted_
```

```
**********************************************************
    1262 Thu Jun  6 20:39:03 2019
new/usr/src/lib/libm/common/R/acosf.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

#endif /* ! codereview */
/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2005 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __acosf = acosf

#include "libm.h"

static const float zero = 0.0f;

float
acosf(float x)
{
acosf(float x) {
        int ix;

        ix = *(int *)&x & ~0x80000000;

#endif /* ! codereview */
        if (ix > 0x3f800000)            /* |x| > 1 or x is nan */
                return ((x * zero) / zero);

#endif /* ! codereview */
        return ((float)acos((double)x));
}
```

```
**********************************************************
    1268 Thu Jun  6 20:39:04 2019
new/usr/src/lib/libm/common/R/acoshf.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

#endif /* ! codereview */
/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2005 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __acoshf = acoshf

#include "libm.h"

static const float zero = 0.0f;

float
acoshf(float x)
{
acoshf(float x) {
        int hx;

        hx = *(int *)&x;

#endif /* ! codereview */
        if (hx < 0x3f800000 || hx > 0x7f800000) /* x < 1 or x is nan */
                return ((x * zero) / zero);

#endif /* ! codereview */
        return ((float)acosh((double)x));
}
```

```
**********************************************************
    1262 Thu Jun  6 20:39:04 2019
new/usr/src/lib/libm/common/R/asinf.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

#endif /* ! codereview */
/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2005 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __asinf = asinf

#include "libm.h"

static const float zero = 0.0f;

float
asinf(float x)
{
asinf(float x) {
        int ix;

        ix = *(int *)&x & ~0x80000000;

#endif /* ! codereview */
        if (ix > 0x3f800000)            /* |x| > 1 or x is nan */
                return ((x * zero) / zero);

#endif /* ! codereview */
        return ((float)asin((double)x));
}
```

```
**********************************************************
    1177 Thu Jun  6 20:39:05 2019
new/usr/src/lib/libm/common/R/asinhf.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
   1 /*
   2  * CDDL HEADER START
   3  *
   4  * The contents of this file are subject to the terms of the
   5  * Common Development and Distribution License (the "License").
   6  * You may not use this file except in compliance with the License.
   7  *
   8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
   9  * or http://www.opensolaris.org/os/licensing.
  10  * See the License for the specific language governing permissions
  11  * and limitations under the License.
  12  *
  13  * When distributing Covered Code, include this CDDL HEADER in each
  14  * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  15  * If applicable, add the following below this CDDL HEADER, with the
  16  * fields enclosed by brackets "[]" replaced with your own identifying
  17  * information: Portions Copyright [yyyy] [name of copyright owner]
  18  *
  19  * CDDL HEADER END
  20  */

  22 /*
  23  * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
  24  */

  26 #endif /* ! codereview */
  27 /*
  28  * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
  29  * Use is subject to license terms.
  30  */

  32 #pragma weak __asinhf = asinhf

  34 #include "libm.h"

  36 float
  37 asinhf(float x)
  38 {
  39         if (isnanf(x))
  25 asinhf(float x) {
  26         if (isnanf(x)) {
  40                 return (x * x);         /* + -> * for Cheetah */
  41         else
  42                 return ((float)asinh((double)x));
  28         } else {
  29                 return ((float) asinh((double) x));
  30         }
  43 }
______unchanged_portion_omitted_
```

```
**********************************************************
    8335 Thu Jun  6 20:39:05 2019
new/usr/src/lib/libm/common/R/atan2f.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

#endif /* ! codereview */
/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2005 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __atan2f = atan2f

#include "libm.h"

#if defined(__i386) && !defined(__amd64)
extern int __swapRP(int);
#endif

/*
 * For i = 0, ..., 192, let x[i] be the double precision number whose
 * high order 32 bits are 0x3f900000 + (i << 16) and whose low order
 * 32 bits are zero.  Then TBL[i] := atan(x[i]) to double precision.
 */

static const double TBL[] = {
        1.56237286204768313e-02,
        1.66000375562312640e-02,
        1.75763148444955872e-02,
        1.85525586258889763e-02,
        1.95287670414137082e-02,
        2.05049382324763683e-02,
        2.14810703409090559e-02,
        2.24571615089905717e-02,
        2.34332098794675855e-02,
        2.44092135955758099e-02,
        2.53851708010611396e-02,
        2.63610796402007873e-02,
        2.73369382578244127e-02,
        2.83127447993351995e-02,
```



```
        2.92884974107309737e-02,
        3.02641942386252458e-02,
        3.12398334302682774e-02,
        3.31909314971115949e-02,
        3.51417768027967800e-02,
        3.70923545503918164e-02,
        3.90426499551669928e-02,
        4.09926482452637811e-02,
        4.29423346623621707e-02,
        4.48916944623464972e-02,
        4.68407129159696539e-02,
        4.87893753095156174e-02,
        5.07376669454602178e-02,
        5.26855731431300420e-02,
        5.46330792393594777e-02,
        5.65801705891457105e-02,
        5.85268325663017702e-02,
        6.04730505641073168e-02,
        6.24188099959573500e-02,
        6.63088949198234884e-02,
        7.01969710718705203e-02,
        7.40829225490337306e-02,
        7.79666338315423008e-02,
        8.18479898030765457e-02,
        8.57268757707448092e-02,
        8.96031774848717461e-02,
        9.34767811585894698e-02,
        9.73475734872236709e-02,
        1.01215441667466668e-01,
        1.05080273416329528e-01,
        1.08941956989865793e-01,
        1.12800381201659389e-01,
        1.16655435441069349e-01,
        1.20507009691224562e-01,
        1.24354994546761438e-01,
        1.32039761614638762e-01,
        1.39708874289163648e-01,
        1.47361481088651630e-01,
        1.54996741923940973e-01,
        1.62613828597948568e-01,
        1.70211925285474408e-01,
        1.77790228992676075e-01,
        1.85347949995694761e-01,
        1.92884312257974672e-01,
        2.00398553825878512e-01,
        2.07889927202262986e-01,
        2.15357699697738048e-01,
        2.22801153759394521e-01,
        2.30219587276843718e-01,
        2.37612313865471242e-01,
        2.44978663126864143e-01,
        2.59629629408257512e-01,
        2.74167451119658789e-01,
        2.88587361894077410e-01,
        3.02884868374971417e-01,
        3.17055753209147029e-01,
        3.31096076704132103e-01,
        3.45002177207105132e-01,
        3.58770670270572245e-01,
        3.72398446676754202e-01,
        3.85882669398073752e-01,
        3.99220769575252543e-01,
        4.12410441597387323e-01,
        4.25449637370042266e-01,
        4.38336559857957830e-01,
        4.51069655988523499e-01,
```

```
        4.63647609000806094e-01,
        4.88333951056405535e-01,
        5.12389460310737732e-01,
        5.35811237960463704e-01,
        5.58599315343562441e-01,
        5.80756353567670414e-01,
        6.02287346134964152e-01,
        6.23199329934065904e-01,
        6.43501108793284371e-01,
        6.63202992706093286e-01,
        6.82316554874748071e-01,
        7.00854407884450192e-01,
        7.18829999621624527e-01,
        7.36257428981428097e-01,
        7.53151280962194414e-01,
        7.69526480405658297e-01,
        7.85398163397448279e-01,
        8.15691923316223422e-01,
        8.44153986113171051e-01,
        8.70903457075652976e-01,
        8.96055384571343927e-01,
        9.19719605350416858e-01,
        9.42000040379463610e-01,
        9.62994330680936206e-01,
        9.82793723247329054e-01,
        1.00148313569423464e+00,
        1.01914134426634972e+00,
        1.03584125300880014e+00,
        1.05165021254837376e+00,
        1.06663036531574362e+00,
        1.08083900054116833e+00,
        1.09432890732118993e+00,
        1.10714871779409041e+00,
        1.13095374397916038e+00,
        1.15257199721566761e+00,
        1.17227388112847630e+00,
        1.19028994968253166e+00,
        1.20681737028525249e+00,
        1.22202532321098967e+00,
        1.23605948947808186e+00,
        1.24904577239825443e+00,
        1.26109338225244039e+00,
        1.27229739520871732e+00,
        1.28274087974427076e+00,
        1.29249666778978534e+00,
        1.30162883400919616e+00,
        1.31019393504755555e+00,
        1.31824205101683711e+00,
        1.32581766366803255e+00,
        1.33970565959899957e+00,
        1.35212738092095464e+00,
        1.36330010035969384e+00,
        1.37340076694501589e+00,
        1.38257482149012589e+00,
        1.39094282700241845e+00,
        1.39860551227195762e+00,
        1.40564764938026987e+00,
        1.41214106460849531e+00,
        1.41814699839963154e+00,
        1.42371797140649403e+00,
        1.42889927219073276e+00,
        1.43373015248470903e+00,
        1.43824479449822262e+00,
        1.44247309910910193e+00,
        1.44644133224813509e+00,
        1.45368758222803240e+00,
```



```
        1.46013910562100091e+00,
        1.46591938806466282e+00,
        1.47112767430373470e+00,
        1.47584462045214027e+00,
        1.48013643959415142e+00,
        1.48405798811891154e+00,
        1.48765509490645531e+00,
        1.49096634108265924e+00,
        1.49402443552511865e+00,
        1.49685728913695626e+00,
        1.49948886200960629e+00,
        1.50193983749385196e+00,
        1.50422816301907281e+00,
        1.50636948736934317e+00,
        1.50837751679893928e+00,
        1.51204050407917401e+00,
        1.51529782154917969e+00,
        1.51821326518395483e+00,
        1.52083793107295384e+00,
        1.52321322351791322e+00,
        1.52537304737331958e+00,
        1.52734543140336587e+00,
        1.52915374769630819e+00,
        1.53081763967160667e+00,
        1.53235373677370856e+00,
        1.53377621092096650e+00,
        1.53509721411557254e+00,
        1.53632722579538861e+00,
        1.53747533091664934e+00,
        1.53854944435964280e+00,
        1.53955649336462841e+00,
        1.54139303859089161e+00,
        1.54302569020147562e+00,
        1.54448660954197448e+00,
        1.54580153317597646e+00,
        1.54699130060982659e+00,
        1.54807296595325550e+00,
        1.54906061995310385e+00,
        1.54996600675867957e+00,
        1.55079899282174605e+00,
        1.55156792769518947e+00,
        1.55227992472688747e+00,
        1.55294108165534417e+00,
        1.55355665560036682e+00,
        1.55413120308095598e+00,
        1.55466869295126031e+00,
        1.55517259817441977e+00,
};

static const double
        pio4 = 7.85398163397448279 00e-01,
        pio2 = 1.57079632679489655 80e+00,
        negpi = -3.14159265358979311 60e+00,
        q1 = -3.33333333333296428 046e-01,
        q2 = 1.99999991868537526 18e-01,
        zero = 0.0;

static const float two24 = 16777216.0;

float
atan2f(float fy, float fx)
{
        double a, t, s, dbase;
        float x, y, base;
        int i, k, hx, hy, ix, iy, sign;
```

```
#endif /* ! codereview */
#if defined(__i386) && !defined(__amd64)
        int rp;
#endif

        iy = *(int *)&fy;
        ix = *(int *)&fx;
        hy = iy & ~0x80000000;
        hx = ix & ~0x80000000;

        sign = 0;

#endif /* ! codereview */
        if (hy > hx) {
                x = fy;
                y = fx;
                i = hx;
                hx = hy;
                hy = i;

#endif /* ! codereview */
                if (iy < 0) {
                        x = -x;
                        sign = 1;
                }

#endif /* ! codereview */
                if (ix < 0) {
                        y = -y;
                        a = pio2;
                } else {
                        a = -pio2;
                        sign = 1 - sign;
                }
        } else {
                y = fy;
                x = fx;

#endif /* ! codereview */
                if (iy < 0) {
                        y = -y;
                        sign = 1;
                }

#endif /* ! codereview */
                if (ix < 0) {
                        x = -x;
                        a = negpi;
                        sign = 1 - sign;
                } else {
                        a = zero;
                }
        }

        if (hx >= 0x7f800000 || hx - hy >= 0x0c800000) {
                if (hx >= 0x7f800000) {
                        if (hx > 0x7f800000)    /* nan */
                                return (x * y);
                        else if (hy >= 0x7f800000)
                                a += pio4;
                } else if ((int)a == 0) {
                        a = (double)y / x;
                }

                return ((float)((sign) ? -a : a));
                return ((float)((sign)? -a : a));
```



```
        }

        if (hy < 0x00800000) {
                if (hy == 0)
                        return ((float)((sign) ? -a : a));

                        return ((float)((sign)? -a : a));
                /* scale subnormal y */
                y *= two24;
                x *= two24;
                hy = *(int *)&y;
                hx = *(int *)&x;
        }

#if defined(__i386) && !defined(__amd64)
        rp = __swapRP(fp_extended);
#endif
        k = (hy - hx + 0x3f800000) & 0xfff80000;

#endif /* ! codereview */
        if (k >= 0x3c800000) {          /* |y/x| >= 1/64 */
                *(int *)&base = k;
                k = (k - 0x3c800000) >> 19;
                a += TBL[k];
        } else {
                /*
                 * For some reason this is faster on USIII than just
                 * doing t = y/x in this case.
                 */
                *(int *)&base = 0;
        }

#endif /* ! codereview */
        dbase = (double)base;
        t = (y - x * dbase) / (x + y * dbase);
        s = t * t;
        a = (a + t) + t * s * (q1 + s * q2);
#if defined(__i386) && !defined(__amd64)
        if (rp != fp_extended)
                (void) __swapRP(rp);
#endif
        return ((float)((sign) ? -a : a));
        return ((float)((sign)? -a : a));
}
```

*_____unchanged_portion_omitted_*

```
**********************************************************
    1480 Thu Jun  6 20:39:06 2019
new/usr/src/lib/libm/common/R/atan2pif.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

#endif /* ! codereview */
/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2005 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#include "libm.h"

static const double invpi = 0.31830988618379067153776752674503;

float
atan2pif(float y, float x)
{
atan2pif(float y, float x) {
        int ix, iy, hx, hy;

        ix = *(int *)&x;
        iy = *(int *)&y;
        hx = ix & ~0x80000000;
        hy = iy & ~0x80000000;

#endif /* ! codereview */
        if (hx > 0x7f800000 || hy > 0x7f800000) /* x or y is nan */
                return (x * y);

#endif /* ! codereview */
        if ((hx | hy) == 0) {
                /* x and y are both zero */
                if (ix == 0)
                        return (y);

                return ((iy == 0) ? 1.0f : -1.0f);
                return ((iy == 0)? 1.0f : -1.0f);
        }
```



```
#endif /* ! codereview */
        return ((float)(invpi * atan2((double)y, (double)x)));
}
```

```
**********************************************************
    5443 Thu Jun  6 20:39:06 2019
new/usr/src/lib/libm/common/R/atanf.c
11210 libm should be cstyle(10NBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __atanf = atanf


/* INDENT OFF */
/*
 * float atanf(float x);
 * Table look-up algorithm
 * By K.C. Ng, March 9, 1989
 *
 * Algorithm.
 *
 * The algorithm is based on atan(x)=atan(y)+atan((x-y)/(1+x*y)).
 * We use poly1(x) to approximate atan(x) for x in [0,1/8] with
 * error (relative)
 *      |(atan(x)-poly1(x))/x|<= 2^-115.94      long double
 *      |(atan(x)-poly1(x))/x|<= 2^-58.85       double
 *      |(atan(x)-poly1(x))/x|<= 2^-25.53       float
 * and use poly2(x) to approximate atan(x) for x in [0,1/65] with
 * error (absolute)
 *      |atan(x)-poly2(x)|<= 2^-122.15          long double
 *      |atan(x)-poly2(x)|<= 2^-64.79           double
 *      |atan(x)-poly2(x)|<= 2^-35.36           float
 * and use poly3(x) to approximate atan(x) for x in [1/8,7/16] with
 * error (relative, on for single precision)
 *      |(atan(x)-poly1(x))/x|<= 2^-25.53       float
 *
 * Here poly1-3 are odd polynomial with the following form:
 *              x + x^3*(a1+x^2*(a2+...))
 *
 * (0). Purge off Inf and NaN and 0
```



```
 * (1). Reduce x to positive by atan(x) = -atan(-x).
 * (2). For x <= 1/8, use
 *      (2.1) if x < 2^(-prec/2-2), atan(x) =  x  with inexact
 *      (2.2) Otherwise
 *              atan(x) = poly1(x)
 * (3). For x >= 8 then
 *      (3.1) if x >= 2^(prec+2),   atan(x) = atan(inf) - pio2lo
 *      (3.2) if x >= 2^(prec/3+2), atan(x) = atan(inf) - 1/x
 *      (3.3) if x >  65,           atan(x) = atan(inf) - poly2(1/x)
 *      (3.4) Otherwise,            atan(x) = atan(inf) - poly1(1/x)
 *
 * (4). Now x is in (0.125, 8)
 *      Find y that match x to 4.5 bit after binary (easy).
 *      If iy is the high word of y, then
 *              single : j = (iy - 0x3e000000) >> 19
 *              (single is modified to (iy-0x3f000000)>>19)
 *              double : j = (iy - 0x3fc00000) >> 16
 *              quad   : j = (iy - 0x3ffc0000) >> 12
 *
 *      Let s = (x-y)/(1+x*y). Then
 *      atan(x) = atan(y) + poly1(s)
 *              = _TBL_r_atan_hi[j] + (_TBL_r_atan_lo[j] + poly2(s) )
 *
 *      Note. |s| <= 1.5384615385e-02 = 1/65. Maxium occurs at x = 1.03125
 *
 */

#include "libm.h"

extern const float _TBL_r_atan_hi[], _TBL_r_atan_lo[];
static const float big = 1.0e37F,
static const float
        big     =   1.0e37F,
        one = 1.0F,
        p1 = -3.333185951111688247225368498733544672172e-0001F,
        p2 = 1.969352894213455405211341983203180636021e-0001F,
        q1 = -3.332921964095646819563419704110132937456e-0001F,
        a1 = -3.333323465223893614063523351509338934592e-0001F,
        a2 = 1.999425625935277805494082274808174062403e-0001F,
        a3 = -1.417547090509737780085769846290301788559e-0001F,
        a4 = 1.016250813871991983097273733227432685084e-0001F,
        a5 = -5.137023693688358515753093811791755221805e-0002F,
        pio2hi = 1.570796371e+0000F,
        pio2lo = -4.371139000e-0008F;

/* INDENT ON */

float
atanf(float xx)
{
atanf(float xx) {
        float x, y, z, r, p, s;
        volatile double dummy __unused;
        int ix, iy, sign, j;

        x = xx;
        ix = *(int *)&x;
        ix = *(int *) &x;
        sign = ix & 0x80000000;
        ix ^= sign;

        /* for |x| < 1/8 */
        if (ix < 0x3e000000) {
                if (ix < 0x38800000) {          /* if |x| < 2**(-prec/2-2) */
                        dummy = big + x;        /* get inexact flag if x != 0 */
#ifdef lint
```

```
                        dummy = dummy;
#endif
                        return (x);
                }

#endif /* ! codereview */
                z = x * x;

#endif /* ! codereview */
                if (ix < 0x3c000000) {  /* if |x| < 2**(-prec/4-1) */
                        x = x + (x * z) * p1;
                        return (x);
                } else {
                        x = x + (x * z) * (p1 + z * p2);
                        return (x);
                }
        }

        /* for |x| >= 8.0 */
        if (ix >= 0x41000000) {
                *(int *)&x = ix;

                *(int *) &x = ix;
                if (ix < 0x42820000) {                          /* x <  65 */
                        r = one / x;
                        z = r * r;
                        y = r * (one + z * (p1 + z * p2));      /* poly1 */
                        y -= pio2lo;
                } else if (ix < 0x44800000) { /* x <  2**(prec/3+2) */
                        r = one / x;
                        z = r * r;
                        y = r * (one + z * q1); /* poly2 */
                        y -= pio2lo;
                } else if (ix < 0x4c800000) { /* x <  2**(prec+2) */
                        y = one / x - pio2lo;
                } else if (ix < 0x7f800000) { /* x <  inf */
                        y = -pio2lo;
                } else {        /* x is inf or NaN */
                        if (ix > 0x7f800000)
                        if (ix > 0x7f800000) {
                                return (x * x); /* - -> * for Cheetah */

                        }
                        y = -pio2lo;
                }

                if (sign == 0)
                        x = pio2hi - y;
                else
                        x = y - pio2hi;

#endif /* ! codereview */
                return (x);
        }


        /* now x is between 1/8 and 8 */
        if (ix < 0x3f000000) {          /* between 1/8 and 1/2 */
                z = x * x;
                x = x + (x * z) * (a1 + z * (a2 + z * (a3 + z * (a4 + z *
                    a5))));
                x = x + (x * z) * (a1 + z * (a2 + z * (a3 + z * (a4 +
                        z * a5))));
                return (x);
        }
```



```
        *(int *)&x = ix;
        *(int *) &x = ix;
        iy = (ix + 0x00040000) & 0x7ff80000;
        *(int *)&y = iy;
        *(int *) &y = iy;
        j = (iy - 0x3f000000) >> 19;

        if (ix == iy) {
        if (ix == iy)
                p = x - y;              /* p=0.0 */
        } else {
        else {
                if (sign == 0)
                        s = (x - y) / (one + x * y);
                else
                        s = (y - x) / (one + x * y);

#endif /* ! codereview */
                z = s * s;
                p = s * (one + z * q1);
        }

#endif /* ! codereview */
        if (sign == 0) {
                r = p + _TBL_r_atan_lo[j];
                x = r + _TBL_r_atan_hi[j];
        } else {
                r = p - _TBL_r_atan_lo[j];
                x = r - _TBL_r_atan_hi[j];
        }

#endif /* ! codereview */
        return (x);
}
```

```
**********************************************************
    1329 Thu Jun  6 20:39:07 2019
new/usr/src/lib/libm/common/R/atanhf.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

#endif /* ! codereview */
/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2005 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __atanhf = atanhf

#include "libm.h"

static const float zero = 0.0f;

float
atanhf(float x)
{
atanhf(float x) {
        int ix;

        ix = *((int *)&x) & ~0x80000000;

#endif /* ! codereview */
        if (ix > 0x3f800000)            /* |x| > 1 or x is nan */
                return ((x * zero) / zero);

#endif /* ! codereview */
        if (ix == 0x3f800000)           /* |x| == 1 */
                return (x / zero);

#endif /* ! codereview */
        return ((float)atanh((double)x));
}
```

```
**********************************************************
   20831 Thu Jun  6 20:39:07 2019
new/usr/src/lib/libm/common/R/besself.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

#endif /* ! codereview */
/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2005 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __j0f = j0f
#pragma weak __j1f = j1f
#pragma weak __jnf = jnf
#pragma weak __y0f = y0f
#pragma weak __y1f = y1f
#pragma weak __ynf = ynf

#include "libm.h"
#include <float.h>

#if defined(__i386) && !defined(__amd64)
extern int __swapRP(int);
#endif

static const float zerof = 0.0f, onef = 1.0f;
static const float
        zerof   = 0.0f,
        onef    = 1.0f;

static const double C[] = {
        0.0,
        -0.125,
        0.25,
        0.375,
        0.5,
        1.0,
        2.0,
        8.0,
        0.564189583547756286948079451560772585844 1,     /* 1/sqrt(pi) */
```



```
        0.636619772367581343075535053490057448,         /* 2/pi */
        1.0e9,
};
______unchanged_portion_omitted_

#define pr              Cj0y0
#define ps              (Cj0y0 + 7)
#define qr              (Cj0y0 + 14)
#define qs              (Cj0y0 + 21)
#define ps      (Cj0y0+7)
#define qr      (Cj0y0+14)
#define qs      (Cj0y0+21)

static const double Cj0[] = {
        -2.500000000000003622131880894830476755537e-0001,       /* r0 */
        1.095597547334830263234433855932375353303e-0002,
        -1.819734750463320921799187258987098087697e-0004,
        9.977001946806131657544212501069893930846e-0007,
        1.0,    /* s0 */
        1.867609810662950169966782360588199673741e-0002,
        1.590389206181565490878430827706972074208e-0004,
        6.520867386742583632375520147714499522721e-0007,
        9.999999999999999942156495584397047660949e-0001,        /* r1 */
        -2.389887722731319130476839836908143731281e-0001,
        1.293359476138939027791270393439493640570e-0002,
        -2.770985642343140122168852400228563364082e-0004,
        2.905241575772067678086738389169625218912e-0006,
        -1.636846356264052597969042009265043251279e-0008,
        5.072306160724884775085431059052611737827e-0011,
        -8.187060730684066824228914775146536139112e-0014,
        5.422219326959949863954297860723723423842e-0017,
        1.0,    /* s1 */
        1.101122772686807702762104741932076228349e-0002,
        6.140169310641649223411427764669143978228e-0005,
        2.292035877515152097976946119293215705250e-0007,
        6.356910426504644334558832036362219583789e-0010,
        1.366626326900219555045096999553948891401e-0012,
        2.280399586866739522891837985560481180088e-0015,
        2.801559820648939665270492520004836611187e-0018,
        2.073101088320349159764410261466350732968e-0021,
};

#define r0              Cj0
#define s0              (Cj0 + 4)
#define r1              (Cj0 + 8)
#define s1              (Cj0 + 17)
#define s0      (Cj0+4)
#define r1      (Cj0+8)
#define s1      (Cj0+17)

static const double Cy0[] = {
        -7.380429510868722526754723020704317641941e-0002,       /* u0 */
        1.772607102684869924301459663049874294814e-0001,
        -1.524370666542713828604078090970799356306e-0002,
        4.650819100693891757143771557629924591915e-0004,
        -7.125768872339528975036316108718239946022e-0006,
        6.411017001656104598327565004771515257146e-0008,
        -3.694275157433032553021246812379258781665e-0010,
        1.434364544206266624252820889648445263842e-0012,
        -3.852064731859936455895036286874139896861e-0015,
        7.182052899726138381739945881914874579696e-0018,
        -9.060556574619677567323741194079797987200e-0021,
        7.124435467408860515265552217131230511455e-0024,
        -2.709726774636397615328813121715432044771e-0027,
        1.0,    /* v0 */
        4.678678931512549002587702477349214886475e-0003,
```

```
        9.486828955529948534822800829497565178985e-0006,
        1.001495929158861646659010844136682454906e-0008,
        4.725338116256021660204443235685358593611e-0012,
};

#define u0              Cy0
#define v0              (Cy0 + 13)
#define v0      (Cy0+13)

static const double Cj1y1[] = {
        -0.443575781679412785711472079 4e7,      /* pr0 */
        -0.994224650507764119565837789 9e7,
        -0.660337324836493910925524543 4e7,
        -0.152352935118113738325510572 2e7,
        -0.109824055434593467273741313 9e6,
        -0.161161664432461011647741289 8e4,
        -0.443575781679412785682801962 7e7,      /* ps0 */
        -0.993412438993458565896755630 9e7,
        -0.658533947972308707282691506 9e7,
        -0.151180950663416088164454635 8e7,
        -0.107263859911038201190306386 7e6,
        -0.145500944019049618245356506 8e4,
        0.332209134098572235185970444 2e5,       /* qr0 */
        0.851451606753357019655500117 1e5,
        0.661788365812708351793999216 6e5,
        0.184942628732238667965200981 9e5,
        0.170637542902076800206128354 6e4,
        0.352651338466360321859217558 0e2,
        0.708712819410287435737750247 2e6,       /* qs0 */
        0.181945804224399729892455383 9e7,
        0.141946066960372089285575525 3e7,
        0.400294435822669751170861081 3e6,
        0.378902297457722026414295225 6e5,
        0.863836776960499096747551718 3e3,
};

#define pr0             Cj1y1
#define ps0             (Cj1y1 + 6)
#define qr0             (Cj1y1 + 12)
#define qs0             (Cj1y1 + 18)
#define ps0     (Cj1y1+6)
#define qr0     (Cj1y1+12)
#define qs0     (Cj1y1+18)

static const double Cj1[] = {
        -6.250000000000022030532009814132189495 48e-0002,        /* a0 */
        1.600998455640072901321605101981501263762e-0003,
        -1.963888815948313758552511884390162864930e-0005,
        8.263917341093549759781339713418201620998e-0008,
        1.0e0,  /* b0 */
        1.605069137643004242395356851797873766927e-0002,
        1.149454623251299996428500249509098499383e-0004,
        3.849701673735260970379681807910852327825e-0007,
        4.999999999999999995517408894340485471724e-0001,
        -6.003825028120475684835384519945468075423e-0002,
        2.301719899263321828388344461995355419832e-0003,
        -4.208494869238892934859525221654040304068e-0005,
        4.377745135188837783031540029700282443388e-0007,
        -2.854106755678624335145364226735677754179e-0009,
        1.234002865443952024332943901323798413689e-0011,
        -3.645498437039791058951273508838177134310e-0014,
        7.404320596071797459925377103787837414422e-0017,
        -1.009457448277522275262808398517024439084e-0019,
        8.520158355824819796968771418801019930585e-0023,
        -3.458159926081163274483854614601091361424e-0026,
        1.0e0,  /* b1 */
```



```
        4.923499437590484879081138588998986303306e-0003,
        1.054389489212184156499666953501976688452e-0005,
        1.180768373106166527048240364872043816050e-0008,
        5.942665743476099355323245707680648588540e-0012,
};

#define a0              Cj1
#define b0              (Cj1 + 4)
#define a1              (Cj1 + 8)
#define b1              (Cj1 + 20)
#define b0      (Cj1+4)
#define a1      (Cj1+8)
#define b1      (Cj1+20)

static const double Cy1[] = {
        -1.960570906462389461018983259589655961560e-0001,        /* c0 */
        4.931824118350661953459180060007970291139e-0002,
        -1.626975871565393656845930125424683008677e-0003,
        1.359657517926394132692884168082224258360e-0005,
        1.0e0,  /* d0 */
        2.565807214838390835108224713630901653793e-0002,
        3.374175208978404268650522752520906231508e-0004,
        2.840368571306070719539936935220728843177e-0006,
        1.396387402048998277638900944415752207592e-0008,
        -1.960570906462389473336339614647555351626e-0001,        /* c1 */
        5.336268030335074494231369159933012844735e-0002,
        -2.684137504382748094149184541866332033280e-0003,
        5.737671618979185736981543498580051903060e-0005,
        -6.642696350686335339171171855576632248 92e-0007,
        4.692417922568160354012347591960362101664e-0009,
        -2.161728635907789319335231338621412258355e-0011,
        6.727353419738316107197644431844194668702e-0014,
        -1.427502986803861372125234355906790573422e-0016,
        2.020392498726806769468143219616642940371e-0019,
        -1.761371948595104156753045457888272716340e-0022,
        7.352828391941157905175042420249225115816e-0026,
        1.0e0,  /* d1 */
        5.029187436727947764916247076102283399442e-0003,
        1.102693095808242775074856548927801750627e-0005,
        1.268035774543174837829534603830227216291e-0008,
        6.579416271766610825192542295821308730206e-0012,
};

#define c0              Cy1
#define d0              (Cy1 + 4)
#define c1              (Cy1 + 9)
#define d1              (Cy1 + 21)
#define d0      (Cy1+4)
#define c1      (Cy1+9)
#define d1      (Cy1+21)

/* core of j0f computation; assumes fx is finite */
static double
__k_j0f(float fx)
{
        double x, z, s, c, ss, cc, r, t, p0, q0;
        int ix, i;

        ix = *(int *)&fx & ~0x80000000;
        x = fabs((double)fx);

#endif /* ! codereview */
        if (ix > 0x41000000) {
                /* x > 8; see comments in j0.c */
                s = sin(x);
```

```
                c = cos(x);

#endif /* ! codereview */
                if (signbit(s) != signbit(c)) {
                        ss = s - c;
                        cc = -cos(x + x) / ss;
                } else {
                        cc = s + c;
                        ss = -cos(x + x) / cc;
                }

#endif /* ! codereview */
                if (ix > 0x501502f9) {
                        /* x > 1.0e10 */
                        p0 = one;
                        q0 = neighth / x;
                } else {
                        t = eight / x;
                        z = t * t;
                        p0 = (pr[0] + z * (pr[1] + z * (pr[2] + z * (pr[3] + z *
                            (pr[4] + z * (pr[5] + z * pr[6])))))) / (ps[0] + z *
                            (ps[1] + z * (ps[2] + z * (ps[3] + z * (ps[4] + z *
                            (ps[5] + z))))));
                        p0 = (pr[0] + z * (pr[1] + z * (pr[2] + z * (pr[3] +
                            z * (pr[4] + z * (pr[5] + z * pr[6])))))) /
                            (ps[0] + z * (ps[1] + z * (ps[2] + z * (ps[3] +
                            z * (ps[4] + z * (ps[5] + z))))));
                        q0 = ((qr[0] + z * (qr[1] + z * (qr[2] + z * (qr[3] +
                            z * (qr[4] + z * (qr[5] + z * qr[6])))))) / (qs[0] +
                            z * (qs[1] + z * (qs[2] + z * (qs[3] + z * (qs[4] +
                            z * (qs[5] + z))))))) * t;
                            z * (qr[4] + z * (qr[5] + z * qr[6])))))) /
                            (qs[0] + z * (qs[1] + z * (qs[2] + z * (qs[3] +
                            z * (qs[4] + z * (qs[5] + z))))))) * t;
                }

#endif /* ! codereview */
                return (isqrtpi * (p0 * cc - q0 * ss) / sqrt(x));
        }

#endif /* ! codereview */
        if (ix <= 0x3727c5ac) {
                /* x <= 1.0e-5 */
                if (ix <= 0x219392ef)   /* x <= 1.0e-18 */
                        return (one - x);

#endif /* ! codereview */
                return (one - x * x * quarter);
        }

#endif /* ! codereview */
        z = x * x;

#endif /* ! codereview */
        if (ix <= 0x3fa3d70a) {
                /* x <= 1.28 */
                r = r0[0] + z * (r0[1] + z * (r0[2] + z * r0[3]));
                s = s0[0] + z * (s0[1] + z * (s0[2] + z * s0[3]));
                return (one + z * (r / s));
        }

#endif /* ! codereview */
        r = r1[8];
        s = s1[8];

#endif /* ! codereview */
```



```
        for (i = 7; i >= 0; i--) {
                r = r * z + r1[i];
                s = s * z + s1[i];
        }

#endif /* ! codereview */
        return (r / s);
}

float
j0f(float fx)
{
        float f;
        int ix;

#endif /* ! codereview */
#if defined(__i386) && !defined(__amd64)
        int rp;
#endif

        ix = *(int *)&fx & ~0x80000000;

#endif /* ! codereview */
        if (ix >= 0x7f800000) {         /* nan or inf */
                if (ix > 0x7f800000)
                        return (fx * fx);

#endif /* ! codereview */
                return (zerof);
        }

#if defined(__i386) && !defined(__amd64)
        rp = __swapRP(fp_extended);
#endif
        f = (float)__k_j0f(fx);
#if defined(__i386) && !defined(__amd64)
        if (rp != fp_extended)
                (void) __swapRP(rp);
#endif
        return (f);
}

/* core of y0f computation; assumes fx is finite and positive */
static double
__k_y0f(float fx)
{
        double x, z, s, c, ss, cc, t, p0, q0, u, v;
        int ix, i;

        ix = *(int *)&fx;
        x = (double)fx;

#endif /* ! codereview */
        if (ix > 0x41000000) {
                /* x > 8; see comments in j0.c */
                s = sin(x);
                c = cos(x);

#endif /* ! codereview */
                if (signbit(s) != signbit(c)) {
                        ss = s - c;
                        cc = -cos(x + x) / ss;
                } else {
                        cc = s + c;
                        ss = -cos(x + x) / cc;
                }
```

```
#endif /* ! codereview */
                if (ix > 0x501502f9) {
                        /* x > 1.0e10 */
                        p0 = one;
                        q0 = neighth / x;
                } else {
                        t = eight / x;
                        z = t * t;
                        p0 = (pr[0] + z * (pr[1] + z * (pr[2] + z * (pr[3] + z *
                            (pr[4] + z * (pr[5] + z * pr[6])))))) / (ps[0] + z *
                            (ps[1] + z * (ps[2] + z * (ps[3] + z * (ps[4] + z *
                            (ps[5] + z))))));
                        p0 = (pr[0] + z * (pr[1] + z * (pr[2] + z * (pr[3] +
                            z * (pr[4] + z * (pr[5] + z * pr[6])))))) /
                            (ps[0] + z * (ps[1] + z * (ps[2] + z * (ps[3] +
                            z * (ps[4] + z * (ps[5] + z))))));
                        q0 = ((qr[0] + z * (qr[1] + z * (qr[2] + z * (qr[3] +
                            z * (qr[4] + z * (qr[5] + z * qr[6])))))) / (qs[0] +
                            z * (qs[1] + z * (qs[2] + z * (qs[3] + z * (qs[4] +
                            z * (qs[5] + z))))))) * t;
                            z * (qr[4] + z * (qr[5] + z * qr[6])))))) /
                            (qs[0] + z * (qs[1] + z * (qs[2] + z * (qs[3] +
                            z * (qs[4] + z * (qs[5] + z))))))) * t;
                }

#endif /* ! codereview */
                return (isqrtpi * (p0 * ss + q0 * cc) / sqrt(x));
        }

#endif /* ! codereview */
        if (ix <= 0x219392ef)           /* x <= 1.0e-18 */
                return (u0[0] + tpi * log(x));

#endif /* ! codereview */
        z = x * x;
        u = u0[12];

#endif /* ! codereview */
        for (i = 11; i >= 0; i--)
                u = u * z + u0[i];

#endif /* ! codereview */
        v = v0[0] + z * (v0[1] + z * (v0[2] + z * (v0[3] + z * v0[4])));
        return (u / v + tpi * (__k_j0f(fx) * log(x)));
}

float
y0f(float fx)
{
        float f;
        int ix;

#endif /* ! codereview */
#if defined(__i386) && !defined(__amd64)
        int rp;
#endif

        ix = *(int *)&fx;

#endif /* ! codereview */
        if ((ix & ~0x80000000) > 0x7f800000)    /* nan */
                return (fx * fx);

#endif /* ! codereview */
        if (ix <= 0) {                          /* zero or negative */
```



```
                if ((ix << 1) == 0)
                        return (-onef / zerof);

#endif /* ! codereview */
                return (zerof / zerof);
        }

#endif /* ! codereview */
        if (ix == 0x7f800000)           /* +inf */
                return (zerof);

#if defined(__i386) && !defined(__amd64)
        rp = __swapRP(fp_extended);
#endif
        f = (float)__k_y0f(fx);
#if defined(__i386) && !defined(__amd64)
        if (rp != fp_extended)
                (void) __swapRP(rp);
#endif
        return (f);
}

/* core of j1f computation; assumes fx is finite */
static double
__k_j1f(float fx)
{
        double x, z, s, c, ss, cc, r, t, p1, q1;
        int i, ix, sgn;

        ix = *(int *)&fx;
        sgn = (unsigned)ix >> 31;
        ix &= ~0x80000000;
        x = fabs((double)fx);

#endif /* ! codereview */
        if (ix > 0x41000000) {
                /* x > 8; see comments in j1.c */
                s = sin(x);
                c = cos(x);

#endif /* ! codereview */
                if (signbit(s) != signbit(c)) {
                        cc = s - c;
                        ss = cos(x + x) / cc;
                } else {
                        ss = -s - c;
                        cc = cos(x + x) / ss;
                }

#endif /* ! codereview */
                if (ix > 0x501502f9) {
                        /* x > 1.0e10 */
                        p1 = one;
                        q1 = three8 / x;
                } else {
                        t = eight / x;
                        z = t * t;
                        p1 = (pr0[0] + z * (pr0[1] + z * (pr0[2] + z * (pr0[3] +
                            z * (pr0[4] + z * pr0[5])))))) / (ps0[0] + z *
                            (ps0[1] + z * (ps0[2] + z * (ps0[3] + z * (ps0[4] +
                            z * (ps0[5] + z))))));
                        p1 = (pr0[0] + z * (pr0[1] + z * (pr0[2] + z *
                            (pr0[3] + z * (pr0[4] + z * pr0[5])))))) /
                            (ps0[0] + z * (ps0[1] + z * (ps0[2] + z *
                            (ps0[3] + z * (ps0[4] + z * (ps0[5] + z))))));
                        q1 = ((qr0[0] + z * (qr0[1] + z * (qr0[2] + z *
```

```
                                (qr0[3] + z * (qr0[4] + z * qr0[5])))))) / (qs0[0] +
                                z * (qs0[1] + z * (qs0[2] + z * (qs0[3] + z *
                                (qs0[4] + z * (qs0[5] + z))))))) * t;
                                (qr0[3] + z * (qr0[4] + z * qr0[5])))))) /
                                (qs0[0] + z * (qs0[1] + z * (qs0[2] + z *
                                (qs0[3] + z * (qs0[4] + z * (qs0[5] + z))))))) * t;
                }

#endif /* ! codereview */
                t = isqrtpi * (p1 * cc - q1 * ss) / sqrt(x);
                return ((sgn) ? -t : t);
                return ((sgn)? -t : t);
        }

#endif /* ! codereview */
        if (ix <= 0x3727c5ac) {
                /* x <= 1.0e-5 */
                if (ix <= 0x219392ef)   /* x <= 1.0e-18 */
                        t = half * x;
                else
                        t = x * (half + neighth * x * x);

                return ((sgn) ? -t : t);
                return ((sgn)? -t : t);
        }

#endif /* ! codereview */
        z = x * x;

#endif /* ! codereview */
        if (ix < 0x3fa3d70a) {
                /* x < 1.28 */
                r = a0[0] + z * (a0[1] + z * (a0[2] + z * a0[3]));
                s = b0[0] + z * (b0[1] + z * (b0[2] + z * b0[3]));
                t = x * half + x * (z * (r / s));
        } else {
                r = a1[11];

#endif /* ! codereview */
                for (i = 10; i >= 0; i--)
                        r = r * z + a1[i];

#endif /* ! codereview */
                s = b1[0] + z * (b1[1] + z * (b1[2] + z * (b1[3] + z * b1[4])));
                t = x * (r / s);
        }

        return ((sgn) ? -t : t);
        return ((sgn)? -t : t);
}

float
j1f(float fx)
{
        float f;
        int ix;

#endif /* ! codereview */
#if defined(__i386) && !defined(__amd64)
        int rp;
#endif

        ix = *(int *)&fx & ~0x80000000;

#endif /* ! codereview */
        if (ix >= 0x7f800000)           /* nan or inf */
```



```
                return (onef / fx);

#if defined(__i386) && !defined(__amd64)
        rp = __swapRP(fp_extended);
#endif
        f = (float)__k_j1f(fx);
#if defined(__i386) && !defined(__amd64)
        if (rp != fp_extended)
                (void) __swapRP(rp);
#endif
        return (f);
}

/* core of y1f computation; assumes fx is finite and positive */
static double
__k_y1f(float fx)
{
        double x, z, s, c, ss, cc, u, v, p1, q1, t;
        int i, ix;

        ix = *(int *)&fx;
        x = (double)fx;

#endif /* ! codereview */
        if (ix > 0x41000000) {
                /* x > 8; see comments in j1.c */
                s = sin(x);
                c = cos(x);

#endif /* ! codereview */
                if (signbit(s) != signbit(c)) {
                        cc = s - c;
                        ss = cos(x + x) / cc;
                } else {
                        ss = -s - c;
                        cc = cos(x + x) / ss;
                }

#endif /* ! codereview */
                if (ix > 0x501502f9) {
                        /* x > 1.0e10 */
                        p1 = one;
                        q1 = three8 / x;
                } else {
                        t = eight / x;
                        z = t * t;
                        p1 = (pr0[0] + z * (pr0[1] + z * (pr0[2] + z * (pr0[3] +
                            z * (pr0[4] + z * pr0[5])))))) / (ps0[0] + z *
                            (ps0[1] + z * (ps0[2] + z * (ps0[3] + z * (ps0[4] +
                            z * (ps0[5] + z))))));
                        p1 = (pr0[0] + z * (pr0[1] + z * (pr0[2] + z *
                            (pr0[3] + z * (pr0[4] + z * pr0[5])))))) /
                            (ps0[0] + z * (ps0[1] + z * (ps0[2] + z *
                            (ps0[3] + z * (ps0[4] + z * (ps0[5] + z))))));
                        q1 = ((qr0[0] + z * (qr0[1] + z * (qr0[2] + z *
                            (qr0[3] + z * (qr0[4] + z * qr0[5])))))) / (qs0[0] +
                            z * (qs0[1] + z * (qs0[2] + z * (qs0[3] + z *
                            (qs0[4] + z * (qs0[5] + z))))))) * t;
                            (qr0[3] + z * (qr0[4] + z * qr0[5])))))) /
                            (qs0[0] + z * (qs0[1] + z * (qs0[2] + z *
                            (qs0[3] + z * (qs0[4] + z * (qs0[5] + z))))))) * t;
                }

#endif /* ! codereview */
                return (isqrtpi * (p1 * ss + q1 * cc) / sqrt(x));
        }
```

```
#endif /* ! codereview */
        if (ix <= 0x219392ef)           /* x <= 1.0e-18 */
                return (-tpi / x);

#endif /* ! codereview */
        z = x * x;

#endif /* ! codereview */
        if (ix < 0x3fa3d70a) {
                /* x < 1.28 */
                u = c0[0] + z * (c0[1] + z * (c0[2] + z * c0[3]));
                v = d0[0] + z * (d0[1] + z * (d0[2] + z * (d0[3] + z * d0[4])));
        } else {
                u = c1[11];

#endif /* ! codereview */
                for (i = 10; i >= 0; i--)
                        u = u * z + c1[i];

#endif /* ! codereview */
                v = d1[0] + z * (d1[1] + z * (d1[2] + z * (d1[3] + z * d1[4])));
        }

#endif /* ! codereview */
        return (x * (u / v) + tpi * (__k_j1f(fx) * log(x) - one / x));
}

float
y1f(float fx)
{
        float f;
        int ix;

#endif /* ! codereview */
#if defined(__i386) && !defined(__amd64)
        int rp;
#endif

        ix = *(int *)&fx;

#endif /* ! codereview */
        if ((ix & ~0x80000000) > 0x7f800000)    /* nan */
                return (fx * fx);

#endif /* ! codereview */
        if (ix <= 0) {                          /* zero or negative */
                if ((ix << 1) == 0)
                        return (-onef / zerof);

#endif /* ! codereview */
                return (zerof / zerof);
        }

#endif /* ! codereview */
        if (ix == 0x7f800000)           /* +inf */
                return (zerof);

#if defined(__i386) && !defined(__amd64)
        rp = __swapRP(fp_extended);
#endif
        f = (float)__k_y1f(fx);
#if defined(__i386) && !defined(__amd64)
        if (rp != fp_extended)
                (void) __swapRP(rp);
#endif
```



```
        return (f);
}

float
jnf(int n, float fx)
{
        double a, b, temp, x, z, w, t, q0, q1, h;
        float f;
        int i, ix, sgn, m, k;

#endif /* ! codereview */
#if defined(__i386) && !defined(__amd64)
        int rp;
#endif

        if (n < 0) {
                n = -n;
                fx = -fx;
        }

#endif /* ! codereview */
        if (n == 0)
                return (j0f(fx));

#endif /* ! codereview */
        if (n == 1)
                return (j1f(fx));

        ix = *(int *)&fx;
        sgn = (n & 1) ? ((unsigned)ix >> 31) : 0;
        sgn = (n & 1)? ((unsigned)ix >> 31) : 0;
        ix &= ~0x80000000;

#endif /* ! codereview */
        if (ix >= 0x7f800000) {         /* nan or inf */
                if (ix > 0x7f800000)
                        return (fx * fx);

                return ((sgn) ? -zerof : zerof);
                return ((sgn)? -zerof : zerof);
        }

#endif /* ! codereview */
        if ((ix << 1) == 0)
                return ((sgn) ? -zerof : zerof);
                return ((sgn)? -zerof : zerof);

#if defined(__i386) && !defined(__amd64)
        rp = __swapRP(fp_extended);
#endif
        fx = fabsf(fx);
        x = (double)fx;

#endif /* ! codereview */
        if ((double)n <= x) {
                /* safe to use J(n+1,x) = 2n/x * J(n,x) - J(n-1,x) */
                a = __k_j0f(fx);
                b = __k_j1f(fx);

#endif /* ! codereview */
                for (i = 1; i < n; i++) {
                        temp = b;
                        b = b * ((double)(i + i) / x) - a;
                        a = temp;
                }
```

```
#endif /* ! codereview */
                f = (float)b;
#if defined(__i386) && !defined(__amd64)
                if (rp != fp_extended)
                        (void) __swapRP(rp);
#endif
                return ((sgn) ? -f : f);
                return ((sgn)? -f : f);
        }

#endif /* ! codereview */
        if (ix < 0x3089705f) {
                /* x < 1.0e-9; use J(n,x) = 1/n! * (x / 2)^n */
                if (n > 6)
                        n = 6;  /* result underflows to zero for n >= 6 */

#endif /* ! codereview */
                b = t = half * x;
                a = one;

#endif /* ! codereview */
                for (i = 2; i <= n; i++) {
                        b *= t;
                        a *= (double)i;
                }

#endif /* ! codereview */
                b /= a;
        } else {
                /* BEGIN CSTYLED */
#endif /* ! codereview */
                /*
                 * Use the backward recurrence:
                 *                        x      x^2      x^2
                 *  J(n,x)/J(n-1,x) =  ---- - ------ - ------   .....
                 *                        2n    2(n+1)   2(n+2)
                 *
                 * Let w = 2n/x and h = 2/x.  Then the above quotient
                 * is equal to the continued fraction:
                 *                    1
                 *      = -----------------------
                 *                        1
                 *         w - -----------------
                 *                       1
                 *              w+h - ---------
                 *                        w+2h - ...
                 *
                 * To determine how many terms are needed, run the
                 * recurrence
                 *
                 *      Q(0) = w,
                 *      Q(1) = w(w+h) - 1,
                 *      Q(k) = (w+k*h)*Q(k-1) - Q(k-2).
                 *
                 * Then when Q(k) > 1e4, k is large enough for single
                 * precision.
                 */
                /* END CSTYLED  */
#endif /* ! codereview */
/* XXX NOT DONE - rework this */
                w = (n + n) / x;
                h = two / x;
                q0 = w;
                z = w + h;
                q1 = w * z - one;
```



```
                k = 1;

#endif /* ! codereview */
                while (q1 < big) {
                        k++;
                        z += h;
                        temp = z * q1 - q0;
                        q0 = q1;
                        q1 = temp;
                }

#endif /* ! codereview */
                m = n + n;
                t = zero;

#endif /* ! codereview */
                for (i = (n + k) << 1; i >= m; i -= 2)
                        t = one / ((double)i / x - t);

#endif /* ! codereview */
                a = t;
                b = one;

#endif /* ! codereview */
                /*
                 * estimate log((2/x)^n*n!) = n*log(2/x)+n*ln(n)
                 * hence, if n*(log(2n/x)) > ...
                 *      single 8.8722839355e+01
                 *      double 7.09782712893383973096e+02
                 *      then recurrent value may overflow and the result is
                 *      likely underflow to zero
                 */
                temp = (double)n;
                temp *= log((two / x) * temp);

#endif /* ! codereview */
                if (temp < 7.09782712893383973096e+02) {
                        for (i = n - 1; i > 0; i--) {
                                temp = b;
                                b = b * ((double)(i + i) / x) - a;
                                a = temp;
                        }
                } else {
                        for (i = n - 1; i > 0; i--) {
                                temp = b;
                                b = b * ((double)(i + i) / x) - a;
                                a = temp;

#endif /* ! codereview */
                                if (b > 1.0e100) {
                                        a /= b;
                                        t /= b;
                                        b = one;
                                }
                        }
                }

#endif /* ! codereview */
                b = (t * __k_j0f(fx) / b);
        }

#endif /* ! codereview */
        f = (float)b;
#if defined(__i386) && !defined(__amd64)
        if (rp != fp_extended)
                (void) __swapRP(rp);
```

```
#endif
        return ((sgn) ? -f : f);
        return ((sgn)? -f : f);
}

float
ynf(int n, float fx)
{
        double a, b, temp, x;
        float f;
        int i, sign, ix;

#endif /* ! codereview */
#if defined(__i386) && !defined(__amd64)
        int rp;
#endif

        sign = 0;

#endif /* ! codereview */
        if (n < 0) {
                n = -n;

#endif /* ! codereview */
                if (n & 1)
                        sign = 1;
        }

#endif /* ! codereview */
        if (n == 0)
                return (y0f(fx));

#endif /* ! codereview */
        if (n == 1)
                return ((sign) ? -y1f(fx) : y1f(fx));
                return ((sign)? -y1f(fx) : y1f(fx));

        ix = *(int *)&fx;

#endif /* ! codereview */
        if ((ix & ~0x80000000) > 0x7f800000)    /* nan */
                return (fx * fx);

#endif /* ! codereview */
        if (ix <= 0) {                          /* zero or negative */
                if ((ix << 1) == 0)
                        return (-onef / zerof);

#endif /* ! codereview */
                return (zerof / zerof);
        }

#endif /* ! codereview */
        if (ix == 0x7f800000)           /* +inf */
                return (zerof);

#if defined(__i386) && !defined(__amd64)
        rp = __swapRP(fp_extended);
#endif
        a = __k_y0f(fx);
        b = __k_y1f(fx);
        x = (double)fx;

#endif /* ! codereview */
        for (i = 1; i < n; i++) {
                temp = b;
```



```
                b *= (double)(i + i) / x;

#endif /* ! codereview */
                if (b <= -DBL_MAX)
                        break;

#endif /* ! codereview */
                b -= a;
                a = temp;
        }

#endif /* ! codereview */
        f = (float)b;
#if defined(__i386) && !defined(__amd64)
        if (rp != fp_extended)
                (void) __swapRP(rp);
#endif
        return ((sign) ? -f : f);
        return ((sign)? -f : f);
}
```

_____*unchanged_portion_omitted_*

```
**********************************************************
    1196 Thu Jun  6 20:39:08 2019
new/usr/src/lib/libm/common/R/cbrtf.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __cbrtf = cbrtf

#include "libm.h"

float
cbrtf(float x)
{
cbrtf(float x) {
#if defined(FPADD_TRAPS_INCOMPLETE_ON_NAN)
        if (isnanf(x))
                return (x * x);
        else
#endif
        return ((float)cbrt((double)x));
        return ((float) cbrt((double) x));
}
______unchanged_portion_omitted_
```

```
**********************************************************
    1187 Thu Jun  6 20:39:08 2019
new/usr/src/lib/libm/common/R/copysignf.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
   1 /*
   2  * CDDL HEADER START
   3  *
   4  * The contents of this file are subject to the terms of the
   5  * Common Development and Distribution License (the "License").
   6  * You may not use this file except in compliance with the License.
   7  *
   8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
   9  * or http://www.opensolaris.org/os/licensing.
  10  * See the License for the specific language governing permissions
  11  * and limitations under the License.
  12  *
  13  * When distributing Covered Code, include this CDDL HEADER in each
  14  * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  15  * If applicable, add the following below this CDDL HEADER, with the
  16  * fields enclosed by brackets "[]" replaced with your own identifying
  17  * information: Portions Copyright [yyyy] [name of copyright owner]
  18  *
  19  * CDDL HEADER END
  20  */

  22 /*
  23  * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
  24  */

  26 #endif /* ! codereview */
  27 /*
  28  * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
  29  * Use is subject to license terms.
  30  */

  32 #pragma weak __copysignf = copysignf

  34 #include "libm.h"

  36 float
  37 copysignf(float x, float y)
  38 {
  25 copysignf(float x, float y) {
  39         float w;

  41         *(int *)&w = (*(int *)&x & ~0x80000000) | (*(int *)&y & 0x80000000);
  28         *(int *) &w = (*(int *) &x & ~0x80000000) | (*(int *) &y & 0x80000000);
  42         return (w);
  43 }
______unchanged_portion_omitted_
```

```
**********************************************************
    3903 Thu Jun  6 20:39:09 2019
new/usr/src/lib/libm/common/R/cosf.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

#endif /* ! codereview */
/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2005 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __cosf = cosf

/*
 * See sincosf.c
 */

#include "libm.h"

extern const int _TBL_ipio2_inf[];
extern int __rem_pio2m(double *, double *, int, int, int, const int *);

#endif /* ! codereview */
#if defined(__i386) && !defined(__amd64)
extern int __swapRP(int);
#endif

static const double C[] = {
        1.85735322054308378716204874632872525989806770558e-0003,
        -1.95035094218403635082921458859320791358115801259e-0004,
        5.38400550766074785970952495168558701485841707252e+0002,
        -3.31975110777873728964197739157371509422022905947e+0001,
        1.09349482127188401868272000389539985058873853699e-0003,
        -5.03324285989964979398034700054920226866107675091e-0004,
        2.43792880266971107750418061559602239831538067410e-0005,
        9.14499072605666582228127405245558035523741471271e+0002,
        -3.63151270591815439197122504991683846785293207730e+0001,
        0.636619772367581343075535,     /* 2^ -1  * 1.45F306DC9C883 */
        0.5,
        1.57079632673412561416 6,       /* 2^  0  * 1.921FB54400000 */
```



```
        6.07710050650619260147 5e-11,    /* 2^-34  * 1.0B4611A626331 */
};

#define S0              C[0]
#define S1              C[1]
#define S2              C[2]
#define S3              C[3]
#define C0              C[4]
#define C1              C[5]
#define C2              C[6]
#define C3              C[7]
#define C4              C[8]
#define invpio2         C[9]
#define half            C[10]
#define pio2_1          C[11]
#define pio2_t          C[12]

float
cosf(float x)
{
        double y, z, w;
        float f;
        int n, ix, hx, hy;
        volatile int i __unused;

        hx = *((int *)&x);
        ix = hx & 0x7fffffff;

        y = (double)x;

        if (ix <= 0x4016cbe4) {                 /* |x| < 3*pi/4 */
                if (ix <= 0x3f490fdb) {         /* |x| < pi/4 */
                        if (ix <= 0x39800000) { /* |x| <= 2**-12 */
                                i = (int)y;
#ifdef lint
                                i = i;
#endif
                                return (1.0f);
                        }

#endif /* ! codereview */
                        z = y * y;
                        return ((float)(((C0 + z * C1) + (z * z) * C2) * (C3 +
                            z * (C4 + z))));
                        return ((float)(((C0 + z * C1) + (z * z) * C2) *
                            (C3 + z * (C4 + z))));
                } else if (hx > 0) {
                        y = (y - pio2_1) - pio2_t;
                        z = y * y;
                        return ((float)-((y * (S0 + z * S1)) * (S2 + z * (S3 +
                            z))));
                        return ((float)-((y * (S0 + z * S1)) *
                            (S2 + z * (S3 + z))));
                } else {
                        y = (y + pio2_1) + pio2_t;
                        z = y * y;
                        return ((float)((y * (S0 + z * S1)) * (S2 + z * (S3 +
                            z))));
                        return ((float)((y * (S0 + z * S1)) *
                            (S2 + z * (S3 + z))));
                }
        } else if (ix <= 0x49c90fdb) {  /* |x| < 2^19*pi */
#if defined(__i386) && !defined(__amd64)
                int rp;

                rp = __swapRP(fp_extended);
```

```
#endif
                w = y * invpio2;

#endif /* ! codereview */
                if (hx < 0)
                        n = (int)(w - half);
                else
                        n = (int)(w + half);

#endif /* ! codereview */
                y = (y - n * pio2_1) - n * pio2_t;
                n++;
#if defined(__i386) && !defined(__amd64)
                if (rp != fp_extended)
                        (void) __swapRP(rp);
#endif
        } else {
                if (ix >= 0x7f800000)
                        return (x / x);         /* cos(Inf or NaN) is NaN */

#endif /* ! codereview */
                hy = ((int *)&y)[HIWORD];
                n = ((hy >> 20) & 0x7ff) - 1046;
                ((int *)&w)[HIWORD] = (hy & 0xfffff) | 0x41600000;
                ((int *)&w)[LOWORD] = ((int *)&y)[LOWORD];
                n = __rem_pio2m(&w, &y, n, 1, 0, _TBL_ipio2_inf) + 1;
        }

        if (n & 1) {
                /* compute cos y */
                z = y * y;
                f = (float)(((C0 + z * C1) + (z * z) * C2) * (C3 + z * (C4 +
                    z)));
                f = (float)(((C0 + z * C1) + (z * z) * C2) *
                    (C3 + z * (C4 + z)));
        } else {
                /* compute sin y */
                z = y * y;
                f = (float)((y * (S0 + z * S1)) * (S2 + z * (S3 + z)));
        }

        return ((n & 2) ? -f : f);
        return ((n & 2)? -f : f);
}
```

_____*unchanged_portion_omitted_*

```
**********************************************************
    1360 Thu Jun  6 20:39:09 2019
new/usr/src/lib/libm/common/R/coshf.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

#endif /* ! codereview */
/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2005 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __coshf = coshf

#include "libm.h"

float
coshf(float x)
{
coshf(float x) {
        double c;
        float w;
        int ix;

        ix = *(int *)&x & ~0x80000000;

#endif /* ! codereview */
        if (ix >= 0x7f800000) {
                /* coshf(x) is |x| if x is +-Inf or NaN */
                return (x * x);
        }

#endif /* ! codereview */
        if (ix >= 0x43000000)           /* coshf(x) trivially overflows */
                c = 1.0e100;
        else
                c = cosh((double)x);

#endif /* ! codereview */
        w = (float)c;
        return (w);
```



```
}
```

```
**********************************************************
    1714 Thu Jun  6 20:39:10 2019
new/usr/src/lib/libm/common/R/erff.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

#endif /* ! codereview */
/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2005 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __erff = erff
#pragma weak __erfcf = erfcf

#include "libm.h"

#if defined(__i386) && !defined(__amd64)
extern int __swapRP(int);
#endif

float
erff(float x)
{
erff(float x) {
        int ix;

        ix = *(int *)&x & ~0x80000000;

#endif /* ! codereview */
        if (ix > 0x7f800000)            /* x is NaN */
                return (x * x);

#endif /* ! codereview */
        return ((float)erf((double)x));
}

float
erfcf(float x)
{
erfcf(float x) {
```



```
        float f;
        int ix;

#endif /* ! codereview */
#if defined(__i386) && !defined(__amd64)
        int rp;
#endif

        ix = *(int *)&x & ~0x80000000;

#endif /* ! codereview */
        if (ix > 0x7f800000)            /* x is NaN */
                return (x * x);

#if defined(__i386) && !defined(__amd64)
        rp = __swapRP(fp_extended);
#endif
        f = (float)erfc((double)x);
#if defined(__i386) && !defined(__amd64)
        if (rp != fp_extended)
                (void) __swapRP(rp);
#endif
        return (f);
}
```

```
**********************************************************
    1198 Thu Jun  6 20:39:10 2019
new/usr/src/lib/libm/common/R/exp10f.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#include "libm.h"

extern double exp10(double);

float
exp10f(float x)
{
exp10f(float x) {
#if defined(FPADD_TRAPS_INCOMPLETE_ON_NAN)
        if (isnanf(x))
                return (x * x);
        else
#endif
        return ((float)exp10((double)x));
        return ((float) exp10((double) x));
}
______unchanged_portion_omitted_
```

```
**********************************************************
    1196 Thu Jun  6 20:39:11 2019
new/usr/src/lib/libm/common/R/exp2f.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
   1 /*
   2  * CDDL HEADER START
   3  *
   4  * The contents of this file are subject to the terms of the
   5  * Common Development and Distribution License (the "License").
   6  * You may not use this file except in compliance with the License.
   7  *
   8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
   9  * or http://www.opensolaris.org/os/licensing.
  10  * See the License for the specific language governing permissions
  11  * and limitations under the License.
  12  *
  13  * When distributing Covered Code, include this CDDL HEADER in each
  14  * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  15  * If applicable, add the following below this CDDL HEADER, with the
  16  * fields enclosed by brackets "[]" replaced with your own identifying
  17  * information: Portions Copyright [yyyy] [name of copyright owner]
  18  *
  19  * CDDL HEADER END
  20  */

  22 /*
  23  * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
  24  */

  26 #endif /* ! codereview */
  27 /*
  28  * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
  29  * Use is subject to license terms.
  30  */

  32 #pragma weak __exp2f = exp2f

  34 #include "libm.h"

  36 float
  37 exp2f(float x)
  38 {
  25 exp2f(float x) {
  39 #if defined(FPADD_TRAPS_INCOMPLETE_ON_NAN)
  40         if (isnanf(x))
  41                 return (x * x);
  42         else
  43 #endif
  44         return ((float)exp2((double)x));
  31         return ((float) exp2((double) x));
  45 }
______unchanged_portion_omitted_
```

```
**********************************************************
   16884 Thu Jun  6 20:39:11 2019
new/usr/src/lib/libm/common/R/expf.c
11210 libm should be cstyle(10NBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

#endif /* ! codereview */
/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __expf = expf


/* INDENT OFF */
/*
 * float expf(float x);
 * Code by K.C. Ng for SUN 5.0 libmopt
 * 11/5/99
 * Method :
 *      1. For |x| >= 2^7, either underflow/overflow.
 *         More precisely:
 *              x > 88.722839355...(0x42B17218) => overflow;
 *              x < -103.97207642..(0xc2CFF1B4) => underflow.
 *      2. For |x| <  2^-6, use polynomail
 *              exp(x) = 1 + x + p1*x^2 + p2*x^3
 *      3. Otherwise, write |x|=(1+r)*2^n, where 0<=r<1.
 *         Let t = 2^n * (1+r) .... x > 0;
 *             t = 2^n * (1-r) .... x < 0. (x= -2**(n+1)+t)
 *         Since -6 <= n <= 6, we may break t into
 *         six 6-bits chunks:
 *                   -5     -11     -17     -23     -29
 *         t=j 1*2+j 2*2  +j 3*2   +j 4*2   +j 5*2   +j 6*2
 *
 *         where 0 <= j i < 64 for i = 1,...,6.
 *
 *         Note that since t has only 24 significant bits,
 *         either j 1 or j 6 must be 0.
```



```
 *                                          7-6i
 *         One may define j i by   (int) ( t * 2     ) mod 64
 *
 *         mathematically. In actual implementation, they can
 *         be obtained by manipulating the exponent and
 *         mantissa bits as follow:
 *              Let ix = (HEX(x)&0x007fffff)|0x00800000.
 *              If n>=0, let ix=ix<<n, then j 6 =0 and
 *                  j i = ix>>(30-6i)) mod 64  ...i=1,...,5
 *              Otherwise, let ix=ix<<(j+6), then j 1 = 0 and
 *                  j i = ix>>(36-6i)) mod 64  ...i=2,...,6
 *
 *      4. Compute exp(t) by table look-up method.
 *         Precompute ET[k] = exp(j*2^(7-6i)), k=j+64*(6-i).
 *         Then
 *         exp(t) = ET[j 1+320]*ET[j 2+256]*ET[j 3+192]*
 *
 *                  ET[j 4+128]*ET[j 5+64]*ET[j 6]
 *
 *                                  n+1
 *      5. If x < 0, return exp(-2   )* exp(t). Note that
 *         -6 <= n <= 6. Let k = n - 6, then we can
 *         precompute
 *                                k-5          n+1
 *         EN[k] = exp(-2   ) = exp(-2   ) for k=0,1,...,12.
 *
 *
 * Special cases:
 *      exp(INF) is INF, exp(NaN) is NaN;
 *      exp(-INF) = 0;
 *      for finite argument, only exp(0) = 1 is exact.
 *
 * Accuracy:
 *      All calculations are done in double precision except for
 *      the case |x| < 2^-6.  When |x| < 2^-6, the error is less
 *      than 0.55 ulp.  When |x| >= 2^-6 and the result is normal,
 *      the error is less than 0.51 ulp.  When FDTOS_TRAPS_... is
 *      defined and the result is subnormal, the error can be as
 *      large as 0.75 ulp.
 */
/* INDENT ON */

#include "libm.h"

/*
 * ET[k] = exp(j*2^(7-6i)) , where j = k mod 64, i = k/64
 */
static const double ET[] = {
        1.0000000000000000000e+00, 1.0000000018626451492 3e+00,
        1.0000000037252902984 6e+00, 1.0000000055879354476 9e+00,
        1.0000000074505805969 2e+00, 1.0000000093132257461 5e+00,
        1.0000000111758708953 9e+00, 1.0000000130385160446 2e+00,
        1.0000000149011611938 5e+00, 1.0000000167638065651 2e+00,
        1.0000000186264517143 5e+00, 1.0000000204890968635 9e+00,
        1.0000000223517420128 2e+00, 1.0000000242143871620 5e+00,
        1.0000000260770325333 2e+00, 1.0000000279396776825 5e+00,
        1.0000000298023228317 8e+00, 1.0000000316649679810 2e+00,
        1.0000000335276133522 9e+00, 1.0000000353902585015 2e+00,
        1.0000000372529036507 5e+00, 1.0000000391155487999 8e+00,
        1.0000000409781941712 6e+00, 1.0000000428408393204 9e+00,
```

```
        1.0000000447034844697 2e+00, 1.0000000465661298410 0e+00,
        1.0000000484287749902 3e+00, 1.0000000502914203615 0e+00,
        1.0000000521540655107 3e+00, 1.0000000540167108820 1e+00,
        1.0000000558793560312 4e+00, 1.0000000577420014025 2e+00,
        1.0000000596046465517 5e+00, 1.0000000614672919230 2e+00,
        1.0000000633299370722 5e+00, 1.0000000651925824435 3e+00,
        1.0000000670552275927 6e+00, 1.0000000689178729640 4e+00,
        1.0000000707805181132 7e+00, 1.0000000726431634845 4e+00,
        1.0000000745058086337 7e+00, 1.0000000763684540050 5e+00,
        1.0000000782310993763 2e+00, 1.0000000800937445255 6e+00,
        1.0000000819563898968 3e+00, 1.0000000838190352681 1e+00,
        1.0000000856816806393 8e+00, 1.0000000875443257886 1e+00,
        1.0000000894069711598 9e+00, 1.0000000912696165311 6e+00,
        1.0000000931322619024 4e+00, 1.0000000949949070516 7e+00,
        1.0000000968575524229 5e+00, 1.0000000987201977942 2e+00,
        1.0000001005828431655 0e+00, 1.0000001024454885367 7e+00,
        1.0000001043081336860 0e+00, 1.0000001061707790572 8e+00,
        1.0000001080334244285 6e+00, 1.0000001098960697898 3e+00,
        1.0000001117587151711 1e+00, 1.0000001136213605423 8e+00,
        1.0000001154840059136 6e+00, 1.0000001173466512849 3e+00,
        1.0000000000000000000 0e+00, 1.0000001192092966562 1e+00,
        1.0000002384186075232 7e+00, 1.0000003576279326011 9e+00,
        1.0000004768372718899 6e+00, 1.0000005960466253895 9e+00,
        1.0000007152559931000 7e+00, 1.0000008344653750214 1e+00,
        1.0000009536747711536 0e+00, 1.0000010728841814966 5e+00,
        1.0000011920936060505 5e+00, 1.0000013113030448153 0e+00,
        1.0000014305124977909 1e+00, 1.0000015497219649773 8e+00,
        1.0000016689314463747 0e+00, 1.0000017881409419828 7e+00,
        1.0000019073504518019 0e+00, 1.0000020265599758317 9e+00,
        1.0000021457695140725 3e+00, 1.0000022649790665241 2e+00,
        1.0000023841886331865 7e+00, 1.0000025033982140598 7e+00,
        1.0000026226078091440 3e+00, 1.0000027418174184390 4e+00,
        1.0000028610270419449 1e+00, 1.0000029802366799776 3e+00,
        1.0000030994463315892 1e+00, 1.0000032186559977276 4e+00,
        1.0000033378656780769 2e+00, 1.0000034570753726370 6e+00,
        1.0000035762850814080 6e+00, 1.0000036954948048899 1e+00,
        1.0000038147045415826 1e+00, 1.0000039339142929861 7e+00,
        1.0000040531240586005 9e+00, 1.0000041723338384258 6e+00,
        1.0000042915436324619 8e+00, 1.0000044107534407089 6e+00,
        1.0000045299632631667 9e+00, 1.0000046491730998354 8e+00,
        1.0000047683829507150 2e+00, 1.0000048875928158054 2e+00,
        1.0000050068026951066 7e+00, 1.0000051260125886187 8e+00,
        1.0000052452224963417 4e+00, 1.0000053644324182755 6e+00,
        1.0000054836423544202 3e+00, 1.0000056028523047757 5e+00,
        1.0000057220622693421 3e+00, 1.0000058412722481193 7e+00,
        1.0000059604822411074 6e+00, 1.0000060796922483064 0e+00,
        1.0000061989022697162 0e+00, 1.0000063181123053368 5e+00,
        1.0000064373223551683 6e+00, 1.0000065565324192107 3e+00,
        1.0000066757424974639 4e+00, 1.0000067949525899280 2e+00,
        1.0000069141626966029 4e+00, 1.0000070333728174887 3e+00,
        1.0000071525829525853 6e+00, 1.0000072717931018928 5e+00,
        1.0000073910032654112 0e+00, 1.0000075102134431404 0e+00,
        1.0000000000000000000 0e+00, 1.0000076294236350804 6e+00,
        1.0000152589054784879 6e+00, 1.0000228884453022251 e+00,
        1.0000305180437909502 4e+00, 1.0000381477002613372 9e+00,
        1.0000457774149413836 5e+00, 1.0000534071878317554 6e+00,
        1.0000610370189331188 6e+00, 1.0000686669082454738 3e+00,
        1.0000762968557694865 3e+00, 1.0000839268615058230 7e+00,
        1.0000915569254544834 6e+00, 1.0000991870476161338 4e+00,
        1.0001068172279914403 3e+00, 1.0001144474665804029 5e+00,
        1.0001220777633836878 1e+00, 1.0001297081184019610 6e+00,
        1.0001373385316352226 9e+00, 1.0001449690030841388 5e+00,
        1.0001525995327493756 5e+00, 1.0001602301206309331 1e+00,
        1.0001678607667294773 6e+00, 1.0001754914710456745 3e+00,
        1.0001831222335795246 2e+00, 1.0001907530543319158 1e+00,
        1.0001983839333028480 9e+00, 1.0002060148704929876 1e+00,
```



```
        1.0002136458659030005 0e+00, 1.0002212769195328867 5e+00,
        1.0002289080313835345 5e+00, 1.0002365392014549438 9e+00,
        1.0002441704297477809 1e+00, 1.0002518017162627117 5e+00,
        1.0002594330609997364 0e+00, 1.0002670644639597430 4e+00,
        1.0002746959251427316 7e+00, 1.0002823274445495904 7e+00,
        1.0002899590221803194 4e+00, 1.0002975906580355847 1e+00,
        1.0003052223521160524 2e+00, 1.0003128541044217225 7e+00,
        1.0003204859149534833 3e+00, 1.0003281177837115567 5e+00,
        1.0003357497106961648 8e+00, 1.0003433816959081958 9e+00,
        1.0003510137393476497 9e+00, 1.0003586458410154147 5e+00,
        1.0003662780009114907 6e+00, 1.0003739102190367660 2e+00,
        1.0003815424953914625 7e+00, 1.0003891748299758024 4e+00,
        1.0003968072227906738 2e+00, 1.0004044396738362987 5e+00,
        1.0004120721831128992 8e+00, 1.0004197047506213635 9e+00,
        1.0004273373763619137 1e+00, 1.0004349700603349937 5e+00,
        1.0004426028025410477 8e+00, 1.0004502356029802978 6e+00,
        1.0004578684616536321 5e+00, 1.0004655013785612727 2e+00,
        1.0004731343537036636 3e+00, 1.0004807673870812490 0e+00,
        1.0000000000000000000 0e+00, 1.0004884004786944728 9e+00,
        1.0009770394924164538 3e+00, 1.0014659171576667517 9e+00,
        1.0019550335910027971 7e+00, 1.0024443889090390857 9e+00,
        1.0029339832284467348 7e+00, 1.0034238166659545932 2e+00,
        1.0039138893383474648 9e+00, 1.0044042013624685516 5e+00,
        1.0048947528552165664 5e+00, 1.0053855439335486199 3e+00,
        1.0058765747144782221 1e+00, 1.0063678453150763925 1e+00,
        1.0068593558524709941 1e+00, 1.0073511064438473994 2e+00,
        1.0078430972064480464 2e+00, 1.0083353282575724385 6e+00,
        1.0088277997145780329 2e+00, 1.0093205116948789079 6e+00,
        1.0098134643159468737 4e+00, 1.0103066576953110278 2e+00,
        1.0108000919505575332 4e+00, 1.0112937671993305066 6e+00,
        1.0117876835593315743 0e+00, 1.0122818411483189837 7e+00,
        1.0127762400841096024 4e+00, 1.0132708804845771410 9e+00,
        1.0137657624676528200 8e+00, 1.0142608861513262574 8e+00,
        1.0147562516536434706 9e+00, 1.0152518590927093189 4e+00,
        1.0157477085866857269 3e+00, 1.0162438002537923509 3e+00,
        1.0167401342123065788 3e+00, 1.0172367105805637521 6e+00,
        1.0177335294769569440 4e+00, 1.0182305910199367371 4e+00,
        1.0187278953280123339 2e+00, 1.0192254425197500022 9e+00,
        1.0197232327137741858 5e+00, 1.0202212660287675039 0e+00,
        1.0207195425834700852 6e+00, 1.0212180624966806785 6e+00,
        1.0217168258872555419 7e+00, 1.0222158328741091093 4e+00,
        1.0227150835762137681 7e+00, 1.0232145781126005257 3e+00,
        1.0237143166023578988 4e+00, 1.0242142991646328020 7e+00,
        1.0247145259186305477 5e+00, 1.0252149969836144016 7e+00,
        1.0257157124789060276 3e+00, 1.0262166725238852649 2e+00,
        1.0267178772379901285 9e+00, 1.0272193267407172534 4e+00,
        1.0277210211516216720 2e+00, 1.0282229605903165925 4e+00,
        1.0287251451764733989 3e+00, 1.0292275750298227610 1e+00,
        1.0297302502701528581 5e+00, 1.0302331710173109335 9e+00,
        1.0307363373912026283 1e+00, 1.0312397495117924251 0e+00,
        1.0000000000000000000 0e+00, 1.0317434074991027603 8e+00,
        1.0644944589178595428 8e+00, 1.0982851403078257579 4e+00,
        1.1331484530668263222 0e+00, 1.1691184461695043328 4e+00,
        1.2062302494209806713 6e+00, 1.2445201077660952293 5e+00,
        1.2840254166877413943 8e+00, 1.3247847587288656967 5e+00,
        1.3668379411737963113 9e+00, 1.4102260349257107474 6e+00,
        1.4549914146182012508 7e+00, 1.5011778000001227972 9e+00,
        1.5488302986341331291 0e+00, 1.5979954499506332510 4e+00,
        1.6487212707001281941 6e+00, 1.7010573018484007601 4e+00,
        1.7550546569602984980 9e+00, 1.8107660721193872266 4e+00,
        1.8682459574322223261 3e+00, 1.9275504501675446711 3e+00,
        1.9887374695822919168 4e+00, 2.0518667734879767472 5e+00,
        2.1170000166126747842 6e+00, 2.1842008108156178991 5e+00,
        2.2535347872132085456 1e+00, 2.3250696602771210308 4e+00,
        2.3988752939670980879 3e+00, 2.4750237699630250887 1e+00,
        2.5535894580629268091 3e+00, 2.6346490888156308685 1e+00,
```

```
        2.71828182845904553488e+00, 2.80456935623722669604e+00,
        2.89359594417176113623e+00, 2.98544853936535581340e+00,
        3.08021684891803104733e+00, 3.17799342753883840018e+00,
        3.27887376793867346692e+00, 3.38295639409246895468e+00,
        3.49034295746184142217e+00, 3.60113833627217561073e+00,
        3.71545073794110392029e+00, 3.83339180475841034834e+00,
        3.95507672292057721464e+00, 4.08062433502646015882e+00,
        4.21015725614395996956e+00, 4.34380199356104235164e+00,
        4.48168907033806451778e+00, 4.62395315278208052234e+00,
        4.77073318196760265408e+00, 4.92217250943229078786e+00,
        5.07841903718008147450e+00, 5.23962536212848917216e+00,
        5.40594892514116676097e+00, 5.57755216479125959239e+00,
        5.75460267600573072144e+00, 5.93727337374560715233e+00,
        6.12574266188198635064e+00, 6.32019460743274397174e+00,
        6.52081912033011246166e+00, 6.72781213889469142941e+00,
        6.94137582119703555605e+00, 7.16171874249371143151e+00,
        1.00000000000000000000e+00, 7.38905609893065040694e+00,
        5.45981500331442362040e+01, 4.03428793492735110249e+02,
        2.98095798704172830185e+03, 2.20264657948067178950e+04,
        1.62754791419003915507e+05, 1.20260428416477679275e+06,
        8.88611052050787210464e+06, 6.56599691373305097222e+07,
        4.85165195409790277481e+08, 3.58491284613159179688e+09,
        2.64891221298434715271e+10, 1.95729609428838775635e+11,
        1.44625706429147509766e+12, 1.06864745815244628906e+13,
        7.89629601826806875000e+13, 5.83461742527454875000e+14,
        4.31123154711519500000e+15, 3.18559317571137560000e+16,
        2.35385266837020000000e+17, 1.73927494152050099200e+18,
        1.28516001143593082880e+19, 9.49611942060244828160e+19,
        7.01673591209763143680e+20, 5.18470552858707204506e+21,
        3.83100800071657691546e+22, 2.83075330327469394756e+23,
        2.09165949601299610311e+24, 1.54553893559010391826e+25,
        1.14200738981568423454e+26, 8.43835666874145383188e+26,
        6.23514908081161674391e+27, 4.60718663433129178064e+28,
        3.40427604993174075827e+29, 2.51543867091916687979e+30,
        1.85867174528412788702e+31, 1.37338297954017610775e+32,
        1.01480038811388874615e+33, 7.49841699699012090701e+33,
        5.54062238439350983445e+34, 4.09399696212745451138e+35,
        3.02507732220114256223e+36, 2.23524660373471497416e+37,
        1.65163625499400180987e+38, 1.22040329431784083418e+39,
        9.01762840503429851945e+39, 6.66317621641089618500e+40,
        4.92345828601205826106e+41, 3.63797094760880474988e+42,
        2.68811714181613560943e+43, 1.98626483613765434356e+44,
        1.46766223015544238535e+45, 1.08446385529002313207e+46,
        8.01316426400059069850e+46, 5.92097202766466993617e+47,
        4.37503944726134096988e+48, 3.23274119108485947460e+49,
        2.38869060142499127023e+50, 1.76501688569176554670e+51,
        1.30418087839363225614e+52, 9.63666567360320166416e+52,
        7.12058632688933793173e+53, 5.26144118266638596909e+54,
};
```

*unchanged_portion_omitted_*

```
#define zero            F[0]
#define one             F[1]
#define p1              F[2]
#define p2              F[3]
#define big             F[4]
#define tiny            F[5]
#if defined(FDTOS_TRAPS_INCOMPLETE_IN_FNS_MODE)
#define twom60          F[6]
#endif

float
expf(float xf)
{
expf(float xf) {
        double w, p, q;
```



```
        int hx, ix, n;

        hx = *(int *)&xf;
        ix = hx & ~0x80000000;

        if (ix < 0x3c800000) {          /* |x| < 2**-6 */
                if (ix < 0x38800000)    /* |x| < 2**-14 */
                        return (one + xf);

#endif /* ! codereview */
                return (one + (xf + (xf * xf) * (p1 + xf * p2)));
        }

        n = ix >> 23;                           /* biased exponent */

        if (n >= 0x86) {                        /* |x| >= 2^7 */
                if (n >= 0xff) {                /* x is nan of +-inf */
                        if (hx == 0xff800000)
                                return (zero);  /* exp(-inf)=0 */

#endif /* ! codereview */
                        return (xf * xf);       /* exp(nan/inf) is nan or inf */
                }

#endif /* ! codereview */
                if (hx > 0)
                        return (big * big);     /* overflow */
                else
                        return (tiny * tiny);   /* underflow */
        }

        ix -= n << 23;

#endif /* ! codereview */
        if (hx > 0)
                ix += 0x800000;
        else
                ix = 0x800000 - ix;

#endif /* ! codereview */
        if (n >= 0x7f) {                /* n >= 0 */
                ix <<= n - 0x7f;
                w = ET[(ix & 0x3f) + 64] * ET[((ix >> 6) & 0x3f) + 128];
                p = ET[((ix >> 12) & 0x3f) + 192] * ET[((ix >> 18) & 0x3f) +
                    256];
                p = ET[((ix >> 12) & 0x3f) + 192] *
                    ET[((ix >> 18) & 0x3f) + 256];
                q = ET[((ix >> 24) & 0x3f) + 320];
        } else {
                ix <<= n - 0x79;
                w = ET[ix & 0x3f] * ET[((ix >> 6) & 0x3f) + 64];
                p = ET[((ix >> 12) & 0x3f) + 128] * ET[((ix >> 18) & 0x3f) +
                    192];
                p = ET[((ix >> 12) & 0x3f) + 128] *
                    ET[((ix >> 18) & 0x3f) + 192];
                q = ET[((ix >> 24) & 0x3f) + 256];
        }

#endif /* ! codereview */
        xf = (float)((w * p) * (hx < 0 ? q * EN[n - 0x79] : q));
#if defined(FDTOS_TRAPS_INCOMPLETE_IN_FNS_MODE)
        if ((unsigned)hx >= 0xc2800000u) {
                if ((unsigned)hx >= 0xc2aeac50) {       /* force underflow */
                        volatile float t = tiny;

#endif /* ! codereview */
```

```
                        t *= t;
                }

#endif /* ! codereview */
                return (xf * twom60);
        }
#endif
        return (xf);
}
```

```
**********************************************************
    1200 Thu Jun  6 20:39:12 2019
new/usr/src/lib/libm/common/R/expm1f.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
   1 /*
   2  * CDDL HEADER START
   3  *
   4  * The contents of this file are subject to the terms of the
   5  * Common Development and Distribution License (the "License").
   6  * You may not use this file except in compliance with the License.
   7  *
   8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
   9  * or http://www.opensolaris.org/os/licensing.
  10  * See the License for the specific language governing permissions
  11  * and limitations under the License.
  12  *
  13  * When distributing Covered Code, include this CDDL HEADER in each
  14  * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  15  * If applicable, add the following below this CDDL HEADER, with the
  16  * fields enclosed by brackets "[]" replaced with your own identifying
  17  * information: Portions Copyright [yyyy] [name of copyright owner]
  18  *
  19  * CDDL HEADER END
  20  */

  22 /*
  23  * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
  24  */

  26 #endif /* ! codereview */
  27 /*
  28  * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
  29  * Use is subject to license terms.
  30  */

  32 #pragma weak __expm1f = expm1f

  34 #include "libm.h"

  36 float
  37 expm1f(float x)
  38 {
  25 expm1f(float x) {
  39 #if defined(FPADD_TRAPS_INCOMPLETE_ON_NAN)
  40         if (isnanf(x))
  41                 return (x * x);
  42         else
  43 #endif
  44         return ((float)expm1((double)x));
  31         return ((float) expm1((double) x));
  45 }
______unchanged_portion_omitted_
```

```
**********************************************************
    1113 Thu Jun  6 20:39:12 2019
new/usr/src/lib/libm/common/R/fabsf.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __fabsf = fabsf

#include "libm.h"

float
fabsf(float x)
{
        *(int *)&x &= ~0x80000000;
fabsf(float x) {
        *(int *) &x &= ~0x80000000;
        return (x);
}
______unchanged_portion_omitted_
```

```
**********************************************************
    2647 Thu Jun  6 20:39:13 2019
new/usr/src/lib/libm/common/R/floorf.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __ceilf = ceilf
#pragma weak __floorf = floorf


/* INDENT OFF */
/*
 * ceilf(x)     return the biggest integral value (in float) below x
 * floorf(x)    return the least integral value (in float) above x
 *
 * NOTE: ceilf(x) and floorf(x) return result
 * with the same sign as x's,  including 0.0F.
 */

#include "libm.h"

static const float xf[] = {
/* ZEROF */
        0.0f,
/* ZEROF */     0.0f,
/* ONEF */ 1.0f,
/* MONEF */ -1.0f,
/* HUGEF */ 1.0e30f, };
/* HUGEF */     1.0e30f,
};

#define ZEROF           xf[0]
#define ONEF            xf[1]
#define MONEF           xf[2]
#define HUGEF           xf[3]
/* INDENT ON */
```



```
float
ceilf(float x)
{
ceilf(float x) {
        volatile float dummy __unused;
        int hx, k, j, ix;

        hx = *(int *)&x;
        hx = *(int *) &x;
        ix = hx & ~0x80000000;
        k = ix >> 23;

#endif /* ! codereview */
        if (((k - 127) ^ (k - 150)) < 0) {
                k = (1 << (150 - k)) - 1;

#endif /* ! codereview */
                if ((k & hx) != 0)
                        dummy = HUGEF + x;      /* raise inexact */

#endif /* ! codereview */
                j = k & (~(hx >> 31));
                *(int *)&x = (hx + j) & ~k;
                *(int *) &x = (hx + j) & ~k;
                return (x);
        } else if (k <= 126) {
                dummy = HUGEF + x;

#endif /* ! codereview */
                if (hx > 0)
                        return (ONEF);
                else if (ix == 0)
                        return (x);
                else
                        return (-ZEROF);
        } else {
        } else
                /* signal invalid if x is a SNaN */
                return (x * ONEF);      /* +0 -> *1 for Cheetah */
        }
#endif /* ! codereview */
}

float
floorf(float x)
{
floorf(float x) {
        volatile float dummy __unused;
        int hx, k, j, ix;

        hx = *(int *)&x;
        hx = *(int *) &x;
        ix = hx & ~0x80000000;
        k = ix >> 23;

#endif /* ! codereview */
        if (((k - 127) ^ (k - 150)) < 0) {
                k = (1 << (150 - k)) - 1;

#endif /* ! codereview */
                if ((k & hx) != 0)
                        dummy = HUGEF + x;      /* raise inexact */

#endif /* ! codereview */
                j = k & (hx >> 31);
```

```
                *(int *)&x = (hx + j) & ~k;
                *(int *) &x = (hx + j) & ~k;
                return (x);
        } else if (k <= 126) {
                dummy = HUGEF + x;

#endif /* ! codereview */
                if (hx > 0)
                        return (ZEROF);
                else if (ix == 0)
                        return (x);
                else
                        return (MONEF);
        } else {
        } else
                /* signal invalid if x is a SNaN */
                return (x * ONEF);      /* +0 -> *1 for Cheetah */
        }
#endif /* ! codereview */
}
```

```
**********************************************************
    3424 Thu Jun  6 20:39:14 2019
new/usr/src/lib/libm/common/R/fmodf.c
11210 libm should be cstyle(10NBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

#endif /* ! codereview */
/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2005 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __fmodf = fmodf

#include "libm.h"

static const int is = (int)0x80000000,
/* INDENT OFF */
static const int
        is = (int)0x80000000,
        im = 0x007fffff,
        ii = 0x7f800000,
        iu = 0x00800000;
/* INDENT ON */

static const float zero = 0.0;

float
fmodf(float x, float y)
{
fmodf(float x, float y) {
        float w;
        int hx, ix, iy, iz, k, ny, nd;

        hx = *(int *)&x;
        ix = hx & 0x7fffffff;
        iy = *(int *)&y & 0x7fffffff;

        /* purge off exception values */
        if (ix >= ii || iy > ii || iy == 0) {
                w = x * y;
```



```
                w = w / w;
        } else if (ix <= iy) {
                if (ix < iy)
                        w = x;          /* return x if |x|<|y| */
                else
                        w = zero * x;   /* return sign(x)*0.0 */
        } else {

                /* INDENT OFF */
                /*
                 * scale x,y to "normal" with
                 *      ny = exponent of y
                 *      nd = exponent of x minus exponent of y
                 */
                /* INDENT ON */
                ny = iy >> 23;
                k = ix >> 23;

                /* special case for subnormal y or x */
                if (ny == 0) {
                        ny = 1;

#endif /* ! codereview */
                        while (iy < iu) {
                                ny -= 1;
                                iy += iy;
                        }

#endif /* ! codereview */
                        nd = k - ny;

#endif /* ! codereview */
                        if (k == 0) {
                                nd += 1;

#endif /* ! codereview */
                                while (ix < iu) {
                                        nd -= 1;
                                        ix += ix;
                                }
                        } else {
                                ix = iu | (ix & im);
                        }
                } else {
                        nd = k - ny;
                        ix = iu | (ix & im);
                        iy = iu | (iy & im);
                }

                /*
                 * fix point fmod for normalized ix and iy
                 */

                /* BEGIN CSTYLED */
                /* fix point fmod for normalized ix and iy */
                /* INDENT OFF */
                /*
                 * while (nd--) {
                 *      iz = ix - iy;
                 * if (iz < 0)
                 *      ix = ix + ix;
                 * else if (iz == 0) {
                 *      *(int *) &w = is & hx;
                 *      return w;
                 * }
                 * else
```

```
 *      ix = iz + iz;
 * }
 */
/* END CSTYLED */

/*
 * unroll the above loop 4 times to gain performance
 */
/* INDENT ON */
/* unroll the above loop 4 times to gain performance */
k = nd >> 2;
nd -= k << 2;

#endif /* ! codereview */
        while (k--) {
                iz = ix - iy;

#endif /* ! codereview */
                if (iz >= 0)
                        ix = iz + iz;
                else
                        ix += ix;

#endif /* ! codereview */
                iz = ix - iy;

#endif /* ! codereview */
                if (iz >= 0)
                        ix = iz + iz;
                else
                        ix += ix;

#endif /* ! codereview */
                iz = ix - iy;

#endif /* ! codereview */
                if (iz >= 0)
                        ix = iz + iz;
                else
                        ix += ix;

#endif /* ! codereview */
                iz = ix - iy;

#endif /* ! codereview */
                if (iz >= 0)
                        ix = iz + iz;
                else
                        ix += ix;

#endif /* ! codereview */
                if (iz == 0) {
                        *(int *)&w = is & hx;
                        return (w);
                }
        }

#endif /* ! codereview */
        while (nd--) {
                iz = ix - iy;

#endif /* ! codereview */
                if (iz >= 0)
                        ix = iz + iz;
                else
                        ix += ix;
```



```
                }

#endif /* ! codereview */
                /* end of unrolling */

                iz = ix - iy;

#endif /* ! codereview */
                if (iz >= 0)
                        ix = iz;

                /* convert back to floating value and restore the sign */
                if (ix == 0) {
                        *(int *)&w = is & hx;
                        return (w);
                }

#endif /* ! codereview */
                while (ix < iu) {
                        ix += ix;
                        ny -= 1;
                }

#endif /* ! codereview */
                while (ix > (iu + iu)) {
                        ny += 1;
                        ix >>= 1;
                }

#endif /* ! codereview */
                if (ny > 0) {
                        *(int *)&w = (is & hx) | (ix & im) | (ny << 23);
                } else {
                        /* subnormal output */
                        k = -ny + 1;
                        ix >>= k;
                        *(int *)&w = (is & hx) | ix;
                }
        }

#endif /* ! codereview */
        return (w);
}
```

```
**********************************************************
    1097 Thu Jun  6 20:39:14 2019
new/usr/src/lib/libm/common/R/gammaf.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

#endif /* ! codereview */
/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2005 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __gammaf = gammaf

#include "libm.h"

float
gammaf(float x)
{
gammaf(float x) {
        return (lgammaf(x));
}
______unchanged_portion_omitted_
```

```
**********************************************************
    1132 Thu Jun  6 20:39:15 2019
new/usr/src/lib/libm/common/R/gammaf_r.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

#endif /* ! codereview */
/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2005 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __gammaf_r = gammaf_r

#include "libm.h"

float
gammaf_r(float x, int *signgamfp)
{
gammaf_r(float x, int *signgamfp) {
        return (lgammaf_r(x, signgamfp));
}
______unchanged_portion_omitted_
```

```
**********************************************************
    1822 Thu Jun  6 20:39:15 2019
new/usr/src/lib/libm/common/R/hypotf.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __hypotf = hypotf

#include "libm.h"

float
hypotf(float x, float y)
{
hypotf(float x, float y) {
        double dx, dy;
        float w;
        int ix, iy;

        ix = (*(int *)&x) & 0x7fffffff;
        iy = (*(int *)&y) & 0x7fffffff;

        ix = (*(int *) &x) & 0x7fffffff;
        iy = (*(int *) &y) & 0x7fffffff;
        if (ix >= 0x7f800000) {
                if (ix == 0x7f800000)
                        *(int *)&w = x == y ? iy : ix; /* w = |x| = inf */
                        *(int *) &w = x == y ? iy : ix; /* w = |x| = inf */
                else if (iy == 0x7f800000)
                        *(int *)&w = x == y ? ix : iy; /* w = |y| = inf */
                        *(int *) &w = x == y ? ix : iy; /* w = |y| = inf */
                else
                        w = fabsf(x) * fabsf(y); /* + -> * for Cheetah */
        } else if (iy >= 0x7f800000) {
                if (iy == 0x7f800000)
                        *(int *)&w = x == y ? ix : iy;  /* w = |y| = inf */
                        *(int *) &w = x == y ? ix : iy; /* w = |y| = inf */
```



```
                else
                        w = fabsf(x) * fabsf(y); /* + -> * for Cheetah */
        } else if (ix == 0) {
                *(int *)&w = iy; /* w = |y|  */
        } else if (iy == 0) {
                *(int *)&w = ix; /* w = |x|  */
        } else {
                dx = (double)x;
                dy = (double)y;
                w = (float)sqrt(dx * dx + dy * dy);
        } else if (ix == 0)
                *(int *) &w = iy;       /* w = |y|  */
        else if (iy == 0)
                *(int *) &w = ix;       /* w = |x|  */
        else {
                dx = (double) x;
                dy = (double) y;
                w = (float) sqrt(dx * dx + dy * dy);
        }

#endif /* ! codereview */
        return (w);
}
```

```
**********************************************************
    2130 Thu Jun  6 20:39:16 2019
new/usr/src/lib/libm/common/R/ilogbf.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __ilogbf = ilogbf

#include "libm.h"
#include "xpg6.h"                       /* __xpg6 */

#if defined(__x86)
static const float two25 = 33554432.0F;
#else
/*
 * v: a non-zero subnormal |x|
 */
static int
ilogbf_subnormal(unsigned v)
{
ilogbf_subnormal(unsigned v) {
        int r = -126 - 23;

        if (v & 0xffff0000)
                r += 16, v >>= 16;

#endif /* ! codereview */
        if (v & 0xff00)
                r += 8, v >>= 8;

#endif /* ! codereview */
        if (v & 0xf0)
                r += 4, v >>= 4;

#endif /* ! codereview */
        v <<= 1;
```



```
        return (r + ((0xffffaa50 >> v) & 0x3));
}
#endif /* defined(__x86) */

static int
raise_invalid(int v)   /* SUSv3 requires ilogbf(0,+/-Inf,NaN) raise invalid */
{
raise_invalid(int v) {  /* SUSv3 requires ilogbf(0,+/-Inf,NaN) raise invalid */
#ifndef lint
        if ((__xpg6 & _C99SUSv3_ilogb_0InfNaN_raises_invalid) != 0) {
                static const double zero = 0.0;
                volatile double dummy __unused;

                dummy = zero / zero;
        }
#endif
        return (v);
}

int
ilogbf(float x)
{
        int k = *((int *)&x) & ~0x80000000;
ilogbf(float x) {
        int k = *((int *) &x) & ~0x80000000;

        if (k < 0x00800000) {
                if (k == 0) {
                if (k == 0)
                        return (raise_invalid(0x80000001));
                } else {
                else {
#if defined(__x86)
                        x *= two25;
                        return (((*((int *)&x) & 0x7f800000) >> 23) - 152);
                        return (((*((int *) &x) & 0x7f800000) >> 23) - 152);
#else
                        return (ilogbf_subnormal(k));
#endif
                }
        } else if (k < 0x7f800000) {
        } else if (k < 0x7f800000)
                return ((k >> 23) - 127);
        } else {
        else
                return (raise_invalid(0x7fffffff));
        }
#endif /* ! codereview */
}
```

```
**********************************************************
    1154 Thu Jun  6 20:39:16 2019
new/usr/src/lib/libm/common/R/isnanf.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __isnanf = isnanf
#pragma weak _isnanf = isnanf

#include "libm.h"

int
isnanf(float x)
{
        return ((*(int *)&x & ~0x80000000) > 0x7f800000);
isnanf(float x) {
        return ((*(int *) &x & ~0x80000000) > 0x7f800000);
}
_____unchanged_portion_omitted_
```

```
**********************************************************
    1270 Thu Jun  6 20:39:17 2019
new/usr/src/lib/libm/common/R/lgammaf.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

#endif /* ! codereview */
/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2005 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __lgammaf = lgammaf

#include "libm.h"

extern int signgamf;

float
lgammaf(float x)
{
lgammaf(float x) {
        float y;

        if (isnanf(x))
                return (x * x);

#endif /* ! codereview */
        y = (float)__k_lgamma((double)x, &signgamf);
        signgam = signgamf;     /* SUSv3 requires the setting of signgam */
        return (y);
}
```

```
**********************************************************
    1186 Thu Jun  6 20:39:17 2019
new/usr/src/lib/libm/common/R/lgammaf_r.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

#endif /* ! codereview */
/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2005 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __lgammaf_r = lgammaf_r

#include "libm.h"

float
lgammaf_r(float x, int *signgamfp)
{
lgammaf_r(float x, int *signgamfp) {
        if (isnanf(x))
                return (x * x);

#endif /* ! codereview */
        return ((float)__k_lgamma((double)x, signgamfp));
}
```

```
**********************************************************
    1416 Thu Jun  6 20:39:18 2019
new/usr/src/lib/libm/common/R/log10f.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

#endif /* ! codereview */
/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2005 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __log10f = log10f

#include "libm.h"

static const float zero = 0.0f, mone = -1.0f;

float
log10f(float x)
{
log10f(float x) {
        int hx, ix;
        float w;

        hx = *(int *)&x;
        ix = hx & ~0x80000000;

#endif /* ! codereview */
        if (ix > 0x7f800000)
                return (x * x);

#endif /* ! codereview */
        if (ix == 0x7f800000)
                return (x + x * x);

#endif /* ! codereview */
        if (ix == 0) {
                w = mone;
                return (w / zero);
        }
```



```
#endif /* ! codereview */
        if (hx < 0) {
                w = zero;
                return (w / zero);
        }

#endif /* ! codereview */
        return ((float)log10((double)x));
}
```

```
**********************************************************
    1423 Thu Jun  6 20:39:18 2019
new/usr/src/lib/libm/common/R/log1pf.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

#endif /* ! codereview */
/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2005 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __log1pf = log1pf

#include "libm.h"

static const float zero = 0.0f;

float
log1pf(float x)
{
log1pf(float x) {
        int ix;

        ix = *(int *)&x;

#endif /* ! codereview */
        if (ix >= 0x7f800000) {
                /* x is +inf or nan */
                return (x * x);
        }

#endif /* ! codereview */
        if (ix < 0) {
                ix &= ~0x80000000;

#endif /* ! codereview */
                if (ix == 0x3f800000)   /* x is -1 */
                        return (x / zero);

#endif /* ! codereview */
```



```
                if (ix > 0x3f800000)    /* x is < -1 or nan */
                        return ((x * zero) / zero);
        }

#endif /* ! codereview */
        return ((float)log1p((double)x));
}
```

```
**********************************************************
    1196 Thu Jun  6 20:39:18 2019
new/usr/src/lib/libm/common/R/log2f.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
   1 /*
   2  * CDDL HEADER START
   3  *
   4  * The contents of this file are subject to the terms of the
   5  * Common Development and Distribution License (the "License").
   6  * You may not use this file except in compliance with the License.
   7  *
   8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
   9  * or http://www.opensolaris.org/os/licensing.
  10  * See the License for the specific language governing permissions
  11  * and limitations under the License.
  12  *
  13  * When distributing Covered Code, include this CDDL HEADER in each
  14  * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  15  * If applicable, add the following below this CDDL HEADER, with the
  16  * fields enclosed by brackets "[]" replaced with your own identifying
  17  * information: Portions Copyright [yyyy] [name of copyright owner]
  18  *
  19  * CDDL HEADER END
  20  */

  22 /*
  23  * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
  24  */

  26 #endif /* ! codereview */
  27 /*
  28  * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
  29  * Use is subject to license terms.
  30  */

  32 #pragma weak __log2f = log2f

  34 #include "libm.h"

  36 float
  37 log2f(float x)
  38 {
  25 log2f(float x) {
  39 #if defined(FPADD_TRAPS_INCOMPLETE_ON_NAN)
  40         if (isnanf(x))
  41                 return (x * x);
  42         else
  43 #endif
  44         return ((float)log2((double)x));
  31         return ((float) log2((double) x));
  45 }
______unchanged_portion_omitted_
```

```
**********************************************************
    2105 Thu Jun  6 20:39:19 2019
new/usr/src/lib/libm/common/R/logbf.c
11210 libm should be cstyle(10NBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __logbf = logbf

#include "libm.h"
#include "xpg6.h"                       /* __xpg6 */
#define _C99SUSv3_logb  _C99SUSv3_logb_subnormal_is_like_ilogb

#if defined(__x86)
static const float two25 = 33554432.0F;
#else
/*
 * v: a non-zero subnormal |x|
 */
static int
ilogbf_subnormal(unsigned v)
{
ilogbf_subnormal(unsigned v) {
        int r = -126 - 23;

        if (v & 0xffff0000)
                r += 16, v >>= 16;

#endif /* ! codereview */
        if (v & 0xff00)
                r += 8, v >>= 8;

#endif /* ! codereview */
        if (v & 0xf0)
                r += 4, v >>= 4;

#endif /* ! codereview */
```



```
        v <<= 1;
        return (r + ((0xffffaa50 >> v) & 0x3));
}
#endif /* defined(__x86) */

static float
raise_division(float t)
{
raise_division(float t) {
#pragma STDC FENV_ACCESS ON

#endif /* ! codereview */
        static const float zero = 0.0F;

#endif /* ! codereview */
        return (t / zero);
}

float
logbf(float x)
{
        int k = *((int *)&x) & ~0x80000000;
logbf(float x) {
        int k = *((int *) &x) & ~0x80000000;

        if (k < 0x00800000) {
                if (k == 0) {
                if (k == 0)
                        return (raise_division(-1.0F));
                } else if ((__xpg6 & _C99SUSv3_logb) != 0) {
                else if ((__xpg6 & _C99SUSv3_logb) != 0) {
#if defined(__x86)
                        x *= two25;
                        return ((float)(((*((int *)&x) & 0x7f800000) >> 23) -
                        return ((float) (((*((int *) &x) & 0x7f800000) >> 23) -
                            152));
#else
                        return ((float)ilogbf_subnormal(k));
                        return ((float) ilogbf_subnormal(k));
#endif
                } else {
                } else
                        return (-126.F);
                }
        } else if (k < 0x7f800000) {
                return ((float)((k >> 23) - 127));
        } else {
        } else if (k < 0x7f800000)
                return ((float) ((k >> 23) - 127));
        else
                return (x * x);
        }
#endif /* ! codereview */
}
```

```
**********************************************************
    4689 Thu Jun  6 20:39:20 2019
new/usr/src/lib/libm/common/R/logf.c
11210 libm should be cstyle(10NBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

#endif /* ! codereview */
/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2005 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __logf = logf

/*
 * Algorithm:
 *
 * Let y = x rounded to six significant bits.  Then for any choice
 * of e and z such that y = 2^e z, we have
 *
 * log(x) = e log(2) + log(z) + log(1+(x-y)/y)
 *
 * Note that (x-y)/y = (x'-y')/y' for any scaled x' = sx, y' = sy;
 * in particular, we can take s to be the power of two that makes
 * ulp(x') = 1.
 *
 * From a table, obtain l = log(z) and r = 1/y'.  For |s| <= 2^-6,
 * approximate log(1+s) by a polynomial p(s) where p(s) := s+s*s*
 * (K1+s*(K2+s*K3)).  Then we compute the expression above as
 * e*ln2 + l + p(r*(x'-y')) all evaluated in double precision.
 *
 * When x is subnormal, we first scale it to the normal range,
 * adjusting e accordingly.
 *
 * Accuracy:
 *
 * The largest error is less than 0.6 ulps.
 */

#include "libm.h"
```



```
/*
 * For i = 0, ..., 12,
 *   TBL[2i] = log(1 + i/32) and TBL[2i+1] = 2^-23 / (1 + i/32)
 *
 * For i = 13, ..., 32,
 *   TBL[2i] = log(1/2 + i/64) and TBL[2i+1] = 2^-23 / (1 + i/32)
 */
static const double TBL[] = {
        0.00000000000000000e+00, 1.192092895507812500e-07,
        3.077165866675368733e-02, 1.155968868371212153e-07,
        6.062462181643483994e-02, 1.121969784007352926e-07,
        8.961215868968713805e-02, 1.089913504464285680e-07,
        1.177830356563834557e-01, 1.059638129340277719e-07,
        1.451820098444978890e-01, 1.030999260979729787e-07,
        1.718502569266592284e-01, 1.003867701480263102e-07,
        1.978257433299198675e-01, 9.781275040064102225e-08,
        2.231435513142097649e-01, 9.536743164062500529e-08,
        2.478361639045812692e-01, 9.304139672256097884e-08,
        2.719337154836417580e-01, 9.082612537202380448e-08,
        2.954642128938358980e-01, 8.871388989825581272e-08,
        3.184537311185345887e-01, 8.669766512784091150e-08,
        -3.522205935893520934e-01, 8.477105034722222546e-08,
        -3.302416868705768671e-01, 8.292820142663043248e-08,
        -3.087354816496132859e-01, 8.116377160904255122e-08,
        -2.876820724517809014e-01, 7.947285970052082892e-08,
        -2.670627852490452536e-01, 7.785096460459183052e-08,
        -2.468600779315257843e-01, 7.629394531250000159e-08,
        -2.270574506353460753e-01, 7.479798560049019504e-08,
        -2.076393647782444896e-01, 7.335956280048077330e-08,
        -1.885911698075500298e-01, 7.197542010613207272e-08,
        -1.698990367953974734e-01, 7.064254195601851460e-08,
        -1.515498981272009327e-01, 6.935813210227272390e-08,
        -1.335313926245226268e-01, 6.811959402901785336e-08,
        -1.158318155251217008e-01, 6.692451343201754014e-08,
        -9.844007281325252434e-02, 6.577064251077586116e-08,
        -8.134563945395240081e-02, 6.465588585805084723e-08,
        -6.453852113757117814e-02, 6.357828776041666578e-08,
        -4.800921918636060631e-02, 6.253602074795082293e-08,
        -3.174869831458029812e-02, 6.152737525201612732e-08,
        -1.574835696813916761e-02, 6.055075024801586965e-08,
        0.00000000000000000e+00, 5.960464477539062500e-08,
};

static const double C[] = {
        6.931471805599452862e-01,
        -2.498875843061889447 06e-01,
        3.333688099812545549 46e-01,
        -5.000000084024749765 65e-01
};

#define ln2             C[0]
#define K3              C[1]
#define K2              C[2]
#define K1              C[3]

float
logf(float x)
{
        double v, t;
        float f;
        int hx, ix, i, exp, iy;

        hx = *(int *)&x;
        ix = hx & ~0x80000000;

        if (ix >= 0x7f800000)           /* nan or inf */
```

```
128                 return ((hx < 0) ? x * 0.0f : x *x);
 21                 return ((hx < 0)? x * 0.0f : x * x);

130         exp = 0;

132 #endif /* ! codereview */
133         if (hx < 0x00800000) {          /* negative, zero, or subnormal */
134                 if (hx <= 0) {
135                         f = 0.0f;
136                         return ((ix == 0) ? -1.0f / f : f / f);
 24                         return ((ix == 0)? -1.0f / f : f / f);
137                 }

139                 /* subnormal; scale by 2^149 */
140                 f = (float)ix;
141                 ix = *(int *)&f;
142                 exp = -149;
143         }

145         exp += (ix - 0x3f320000) >> 23;
146         ix &= 0x007fffff;
147         iy = (ix + 0x20000) & 0xfffc0000;
148         i = iy >> 17;
149         t = ln2 * (double)exp + TBL[i];
150         v = (double)(ix - iy) * TBL[i + 1];
151         v += (v * v) * (K1 + v * (K2 + v * K3));
152         f = (float)(t + v);
153         return (f);
154 }
```

*_____unchanged_portion_omitted_*

```
**********************************************************
    1944 Thu Jun  6 20:39:20 2019
new/usr/src/lib/libm/common/R/nextafterf.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
   1 /*
   2  * CDDL HEADER START
   3  *
   4  * The contents of this file are subject to the terms of the
   5  * Common Development and Distribution License (the "License").
   6  * You may not use this file except in compliance with the License.
   7  *
   8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
   9  * or http://www.opensolaris.org/os/licensing.
  10  * See the License for the specific language governing permissions
  11  * and limitations under the License.
  12  *
  13  * When distributing Covered Code, include this CDDL HEADER in each
  14  * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  15  * If applicable, add the following below this CDDL HEADER, with the
  16  * fields enclosed by brackets "[]" replaced with your own identifying
  17  * information: Portions Copyright [yyyy] [name of copyright owner]
  18  *
  19  * CDDL HEADER END
  20  */

  22 /*
  23  * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
  24  */

  26 #endif /* ! codereview */
  27 /*
  28  * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
  29  * Use is subject to license terms.
  30  */

  32 #pragma weak __nextafterf = nextafterf

  34 #include "libm.h"

  36 float
  37 nextafterf(float x, float y)
  38 {
  25 nextafterf(float x, float y) {
  39         float w;
  40         int *pw = (int *)&w;
  41         int *px = (int *)&x;
  42         int *py = (int *)&y;
  27         int *pw = (int *) &w;
  28         int *px = (int *) &x;
  29         int *py = (int *) &y;
  43         int ix, iy, iz;

  45         ix = px[0];
  46         iy = py[0];

  48 #endif /* ! codereview */
  49         if ((ix & ~0x80000000) > 0x7f800000)
  50                 return (x * y);         /* + -> * for Cheetah */

  52 #endif /* ! codereview */
  53         if ((iy & ~0x80000000) > 0x7f800000)
  54                 return (y * x);         /* + -> * for Cheetah */

  56 #endif /* ! codereview */
  57         if (ix == iy || (ix | iy) == 0x80000000)
```



```
  58                 return (y);             /* C99 requirement */

  60         if ((ix & ~0x80000000) == 0) {
  34         if ((ix & ~0x80000000) == 0)
  61                 iz = 1 | (iy & 0x80000000);
  62         } else if (ix > 0) {
  36         else if (ix > 0) {
  63                 if (ix > iy)
  64                         iz = ix - 1;
  65                 else
  66                         iz = ix + 1;
  67         } else {
  68                 if (iy < 0 && ix < iy)
  69                         iz = ix + 1;
  70                 else
  71                         iz = ix - 1;
  72         }

  74 #endif /* ! codereview */
  75         pw[0] = iz;
  76         ix = iz & 0x7f800000;

  78 #endif /* ! codereview */
  79         if (ix == 0x7f800000) {
  80                 /* raise overflow */
  81                 volatile float t;

  83                 *(int *)&t = 0x7f7fffff;
  47                 *(int *) &t = 0x7f7fffff;
  84                 t *= t;
  85         } else if (ix == 0) {
  86                 /* raise underflow */
  87                 volatile float t;

  89                 *(int *)&t = 0x00800000;
  53                 *(int *) &t = 0x00800000;
  90                 t *= t;
  91         }

  93 #endif /* ! codereview */
  94         return (w);
  95 }
```

```
**********************************************************
    8045 Thu Jun  6 20:39:21 2019
new/usr/src/lib/libm/common/R/powf.c
11210 libm should be cstyle(10NBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

#endif /* ! codereview */
/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2005 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __powf = powf

#include "libm.h"
#include "xpg6.h"                       /* __xpg6 */
#define _C99SUSv3_pow   _C99SUSv3_pow_treats_Inf_as_an_even_int

#if defined(__i386) && !defined(__amd64)
extern int __swapRP(int);
#endif

/* INDENT OFF */
static const double
        ln2 = 6.9314718055994528622 7e-01,    /* 0x3fe62e42, 0xfefa39ef */
        invln2 = 1.4426950408889633870 0e+00, /* 0x3ff71547, 0x652b82fe */
        dtwo = 2.0,
        done = 1.0,
        dhalf = 0.5,
        d32 = 32.0,
        d1_32 = 0.03125,
        A0 = 1.9999999998137233036475111469959664392 50e+0000,
        A1 = 6.6669108179358585337701386571396656086 10e-0001,
        t0 = 2.0000000000047774892624053150732037469 43e+0000,
        t1 = 1.6666634083499263798731119329942507267 07e-0001;

static const double S[] = {
        1.0000000000000000000e+00,     /* 3FF0000000000000 */
        1.0218971486541166271 4e+00,     /* 3FF059B0D3158574 */
        1.0442737824274137548 0e+00,     /* 3FF0B5586CF9890F */
        1.0671404006768236971 7e+00,     /* 3FF11301D0125B51 */
```



```
        1.0905077326652576896 7e+00,     /* 3FF172B83C7D517B */
        1.1143867425958924322 1e+00,     /* 3FF1D4873168B9AA */
        1.1387886347566915645 8e+00,     /* 3FF2387A6E756238 */
        1.1637248587775774755 2e+00,     /* 3FF29E9DF51FDEE1 */
        1.1892071150027210269 0e+00,     /* 3FF306FE0A31B715 */
        1.2152473599804689552 4e+00,     /* 3FF371A7373AA9CB */
        1.2418578120734840020 1e+00,     /* 3FF3DEA64C123422 */
        1.2690509571917332198 9e+00,     /* 3FF44E086061892D */
        1.2968395546510096405 5e+00,     /* 3FF4BFDAD5362A27 */
        1.3252366431597413232 2e+00,     /* 3FF5342B569D4F82 */
        1.3542555469368926512 9e+00,     /* 3FF5AB07DD485429 */
        1.3839098819638320225 8e+00,     /* 3FF6247EB03A5585 */
        1.4142135623730951454 7e+00,     /* 3FF6A09E667F3BCD */
        1.4451808069770466502 7e+00,     /* 3FF71F75E8EC5F74 */
        1.4768261459394993462 3e+00,     /* 3FF7A11473EB0187 */
        1.5091644275934228414 1e+00,     /* 3FF82589994CCE13 */
        1.5422108254079407441 1e+00,     /* 3FF8ACE5422AA0DB */
        1.5759808451078864965 9e+00,     /* 3FF93737B0CDC5E5 */
        1.6104903319492542834 7e+00,     /* 3FF9C49182A3F090 */
        1.6457554781539649457 8e+00,     /* 3FFA5503B23E255D */
        1.6817928305074290040 7e+00,     /* 3FFAE89F995AD3AD */
        1.7186192981224779341 4e+00,     /* 3FFB7F76F2FB5E47 */
        1.7562521603732994535 1e+00,     /* 3FFC199BDD85529C */
        1.7947090750031071682 0e+00,     /* 3FFCB720DCEF9069 */
        1.8340080864093424306 6e+00,     /* 3FFD5818DCFBA487 */
        1.8741676341102999625 6e+00,     /* 3FFDFC97337B9B5F */
        1.9152065613971474000 7e+00,     /* 3FFEA4AFA2A490DA */
        1.9571441241754001794 1e+00,     /* 3FFF50765B6E4540 */
};

static const double TBL[] = {
        0.0000000000000000000e+00, 3.0771658666753687 3e-02,
        6.0624621816434839 9e-02, 8.9612158689687138 0e-02,
        1.1778303565638345 6e-01, 1.4518200984449788 9e-01,
        1.7185025692665922 8e-01, 1.9782574332991986 8e-01,
        2.2314355131420976 5e-01, 2.4783616390458126 9e-01,
        2.7193371548364175 8e-01, 2.9546421289383589 8e-01,
        3.1845373111853458 9e-01, 3.4092658697059319 3e-01,
        3.6290549368936847 5e-01, 3.8441169891033205 6e-01,
        4.0546510810816438 5e-01, 4.2608439531090008 8e-01,
        4.4628710262841953 0e-01, 4.6608972992459923 9e-01,
        4.8550781578170082 4e-01, 5.0455601075239531 2e-01,
        5.2324814376454786 8e-01, 5.4159728243274440 9e-01,
        5.5961578793542265 9e-01, 5.7731536503482361 3e-01,
        5.9470710774669277 6e-01, 6.1180154110599294 1e-01,
        6.2860865942237409 4e-01, 6.4513796137358470 1e-01,
        6.6139848224536501 6e-01, 6.7739882359180614 3e-01,
        0.0000000000000000e+00,
        3.0771658666753687 3e-02,
        6.0624621816434839 9e-02,
        8.9612158689687138 0e-02,
        1.1778303565638345 6e-01,
        1.4518200984449788 9e-01,
        1.7185025692665922 8e-01,
        1.9782574332991986 8e-01,
        2.2314355131420976 5e-01,
        2.4783616390458126 9e-01,
        2.7193371548364175 8e-01,
        2.9546421289383589 8e-01,
        3.1845373111853458 9e-01,
        3.4092658697059319 3e-01,
        3.6290549368936847 5e-01,
        3.8441169891033205 6e-01,
        4.0546510810816438 5e-01,
        4.2608439531090008 8e-01,
        4.4628710262841953 0e-01,
```

```
        4.66089729924599239e-01,
        4.85507815781700824e-01,
        5.04556010752395312e-01,
        5.23248143764547868e-01,
        5.41597282432744409e-01,
        5.59615787935422659e-01,
        5.77315365034823613e-01,
        5.94707107746692776e-01,
        6.11801541105992941e-01,
        6.28608659422374094e-01,
        6.45137961373584701e-01,
        6.61398482245365016e-01,
        6.77398823591806143e-01,
};

static const float zero = 0.0F, one = 1.0F, huge = 1.0e25f, tiny = 1.0e-25f;

/* INDENT ON */

float
powf(float x, float y)
{
powf(float x, float y) {
        float fx = x, fy = y;
        float fz;
        int ix, iy, jx, jy, k, iw, yisint;

        ix = *(int *)&x;
        iy = *(int *)&y;
        jx = ix & ~0x80000000;
        jy = iy & ~0x80000000;

        if (jy == 0)
                return (one);           /* x**+-0 = 1 */
        else if (ix == 0x3f800000 && (__xpg6 & _C99SUSv3_pow) != 0)
                return (one);           /* C99: 1**anything = 1 */
        else if (((0x7f800000 - jx) | (0x7f800000 - jy)) < 0)
                return (fx * fy);       /* at least one of x or y is NaN */

        /*
         * includes Sun: 1**NaN = NaN
         */

                                        /* includes Sun: 1**NaN = NaN */
        /* INDENT OFF */
        /*
         * determine if y is an odd int
         * yisint = 0 ... y is not an integer
         * yisint = 1 ... y is an odd int
         * yisint = 2 ... y is an even int
         */
        /* INDENT ON */
        yisint = 0;

#endif /* ! codereview */
        if (ix < 0) {
                if (jy >= 0x4b800000) {
                        yisint = 2;             /* |y|>=2**24: y must be even */
                } else if (jy >= 0x3f800000) {
                        k = (jy >> 23) - 0x7f;  /* exponent */
                        iw = jy >> (23 - k);

#endif /* ! codereview */
                        if ((iw << (23 - k)) == jy)
                                yisint = 2 - (iw & 1);
                }
```



```
        }

        /* special value of y */
        if ((jy & ~0x7f800000) == 0) {
                if (jy == 0x7f800000) { /* y is +-inf */
                        if (jx == 0x3f800000) {
                                if ((__xpg6 & _C99SUSv3_pow) != 0)
                                        fz = one;
                                /* C99: (-1)**+-inf is 1 */
                                else
                                        fz = fy - fy;

#endif /* ! codereview */
                                /* Sun: (+-1)**+-inf = NaN */
                        } else if (jx > 0x3f800000) {
                                /* (|x|>1)**+,-inf = inf,0 */
                                if (iy > 0)
                                        fz = fy;
                                else
                                        fz = zero;
                        } else {        /* (|x|<1)**-,+inf = inf,0 */
                                if (iy < 0)
                                        fz = -fy;
                                else
                                        fz = zero;
                        }

#endif /* ! codereview */
                        return (fz);
                } else if (jy == 0x3f800000) {  /* y is +-1 */
                        if (iy < 0)
                                fx = one / fx;  /* y is -1 */

#endif /* ! codereview */
                        return (fx);
                } else if (iy == 0x40000000) {  /* y is 2 */
                        return (fx * fx);
                } else if (iy == 0x3f000000) {  /* y is 0.5 */
                        if (jx != 0 && jx != 0x7f800000)
                                return (sqrtf(x));
                }
        }

        /* special value of x */
        if ((jx & ~0x7f800000) == 0) {
                if (jx == 0x7f800000 || jx == 0 || jx == 0x3f800000) {
                        /* x is +-0,+-inf,-1; set fz = |x|**y */
                        *(int *)&fz = jx;

#endif /* ! codereview */
                        if (iy < 0)
                                fz = one / fz;

#endif /* ! codereview */
                        if (ix < 0) {
                                if (jx == 0x3f800000 && yisint == 0) {
                                        /* (-1)**non-int is NaN */
                                        fz = zero;
                                        fz /= fz;
                                } else if (yisint == 1) {
                                        /* (x<0)**odd = -(|x|**odd) */
                                        fz = -fz;
                                }
                        }

#endif /* ! codereview */
```

```
                        return (fz);
                }
        }

        /* (x<0)**(non-int) is NaN */
        if (ix < 0 && yisint == 0) {
                fz = zero;
                return (fz / fz);
        }

        /*
         * compute exp(y*log(|x|))
         * fx = *(float *) &jx;
         * fz = (float) exp(((double) fy) * log((double) fx));
         */
        {
                double dx, dy, dz, ds;
                int *px = (int *)&dx, *pz = (int *)&dz, i, n, m;

#endif /* ! codereview */
#if defined(__i386) && !defined(__amd64)
                int rp = __swapRP(fp_extended);
#endif

                fx = *(float *)&jx;
                dx = (double)fx;

                /* compute log(x)/ln2 */
                i = px[HIWORD] + 0x4000;
                n = (i >> 20) - 0x3ff;
                pz[HIWORD] = i & 0xffff8000;
                pz[LOWORD] = 0;
                ds = (dx - dz) / (dx + dz);
                i = (i >> 15) & 0x1f;
                dz = ds * ds;
                dy = invln2 * (TBL[i] + ds * (A0 + dz * A1));

#endif /* ! codereview */
                if (n == 0)
                        dz = (double)fy * dy;
                else
                        dz = (double)fy * (dy + (double)n);

                /* compute exp2(dz=y*ln(x)) */
                i = pz[HIWORD];

#endif /* ! codereview */
                if ((i & ~0x80000000) >= 0x40640000) {  /* |z| >= 160.0 */
                        fz = (i > 0) ? huge : tiny;

                        fz = (i > 0)? huge : tiny;
                        if (ix < 0 && yisint == 1)
                                fz *= -fz;      /* (-ve)**(odd int) */
                        else
                                fz *= fz;

#endif /* ! codereview */
#if defined(__i386) && !defined(__amd64)
                        if (rp != fp_extended)
                                (void) __swapRP(rp);
#endif
                        return (fz);
                }

                n = (int)(d32 * dz + (i > 0 ? dhalf : -dhalf));
                i = n & 0x1f;
```



```
                m = n >> 5;
                dy = ln2 * (dz - d1_32 * (double)n);
                dx = S[i] * (done - (dtwo * dy) / (dy * (done - dy * t1) - t0));

#endif /* ! codereview */
                if (m != 0)
                        px[HIWORD] += m << 20;

#endif /* ! codereview */
                fz = (float)dx;
#if defined(__i386) && !defined(__amd64)
                if (rp != fp_extended)
                        (void) __swapRP(rp);
#endif
        }

        /* end of computing exp(y*log(x)) */
        if (ix < 0 && yisint == 1)
                fz = -fz;               /* (-ve)**(odd int) */

#endif /* ! codereview */
        return (fz);
}
```

```
**********************************************************
    1379 Thu Jun  6 20:39:21 2019
new/usr/src/lib/libm/common/R/remainderf.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
   1 /*
   2  * CDDL HEADER START
   3  *
   4  * The contents of this file are subject to the terms of the
   5  * Common Development and Distribution License (the "License").
   6  * You may not use this file except in compliance with the License.
   7  *
   8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
   9  * or http://www.opensolaris.org/os/licensing.
  10  * See the License for the specific language governing permissions
  11  * and limitations under the License.
  12  *
  13  * When distributing Covered Code, include this CDDL HEADER in each
  14  * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  15  * If applicable, add the following below this CDDL HEADER, with the
  16  * fields enclosed by brackets "[]" replaced with your own identifying
  17  * information: Portions Copyright [yyyy] [name of copyright owner]
  18  *
  19  * CDDL HEADER END
  20  */

  22 /*
  23  * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
  24  */

  26 #endif /* ! codereview */
  27 /*
  28  * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
  29  * Use is subject to license terms.
  30  */

  32 #pragma weak __remainderf = remainderf

  34 #include "libm.h"

  36 float
  37 remainderf(float x, float y)
  38 {
  25 remainderf(float x, float y) {
  39         if (isnanf(x) || isnanf(y))
  40                 return (x * y);

  42         if (y == 0.0f || (*(int *)&x & ~0x80000000) == 0x7f800000) {
  28         if (y == 0.0f || (*(int *) &x & ~0x80000000) == 0x7f800000) {
  43                 /* y is 0 or x is infinite; raise invalid and return NaN */
  44                 y = 0.0f;
  45                 *(int *)&x = 0x7f800000;
  31                 *(int *) &x = 0x7f800000;
  46                 return (x * y);
  47         }

  49         return ((float)remainder((double)x, (double)y));
  34         return ((float) remainder((double) x, (double) y));
  50 }
______unchanged_portion_omitted_
```

```
**********************************************************
    3661 Thu Jun  6 20:39:21 2019
new/usr/src/lib/libm/common/R/rintf.c
11210 libm should be cstyle(10NBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __rintf = rintf


/* INDENT OFF */
/*
 * aintf(x)     return x chopped to integral value
 * anintf(x)    return sign(x)*(|x|+0.5) chopped to integral value
 * irintf(x)    return rint(x) in integer format
 * nintf(x)     return anint(x) in integer format
 * rintf(x)     return x rounded to integral according to the rounding direction
 *
 * NOTE: rintf(x), aintf(x) and anintf(x) return results with the same sign as
 * x's,  including 0.0.
 */

#include "libm.h"

static const float xf[] = {
/* ZEROF */
        0.0f,
/* ZEROF */     0.0f,
/* TWO_23F */ 8.3886080000e6f,
/* MTWO_23F */ -8.3886080000e6f,
/* ONEF */ 1.0f,
/* MONEF */ -1.0f,
/* HALFF */ 0.5f,
/* MHALFF */ -0.5f,
/* HUGEF */ 1.0e30f,
};
```



```
#define ZEROF           xf[0]
#define TWO_23F         xf[1]
#define MTWO_23F        xf[2]
#define ONEF            xf[3]
#define MONEF           xf[4]
#define HALFF           xf[5]
#define MHALFF          xf[6]
#define HUGEF           xf[7]
/* INDENT ON */

float
aintf(float x)
{
aintf(float x) {
        int hx, k;
        float y;

        hx = *(int *)&x;
        hx = *(int *) &x;
        k = (hx & ~0x80000000) >> 23;

#endif /* ! codereview */
        if (k < 150) {
                y = (float)((int)x);

                y = (float) ((int) x);
                /*
                 * make sure y has the same sign of x when |x|<0.5
                 * (i.e., y=0.0)
                 */
                return (((k - 127) & hx) < 0 ? -y : y);
        } else {
        } else
                /* signal invalid if x is a SNaN */
                return (x * ONEF);      /* +0 -> *1 for Cheetah */
        }
#endif /* ! codereview */
}

float
anintf(float x)
{
anintf(float x) {
        volatile float dummy __unused;
        int hx, k, j, ix;

        hx = *(int *)&x;
        hx = *(int *) &x;
        ix = hx & ~0x80000000;
        k = ix >> 23;

#endif /* ! codereview */
        if (((k - 127) ^ (k - 150)) < 0) {
                j = 1 << (149 - k);
                k = j + j - 1;

#endif /* ! codereview */
                if ((k & hx) != 0)
                        dummy = HUGEF + x;      /* raise inexact */

                *(int *)&x = (hx + j) & ~k;
                *(int *) &x = (hx + j) & ~k;
                return (x);
        } else if (k <= 126) {
                dummy = HUGEF + x;
                *(int *)&x = (0x3f800000 & ((125 - k) >> 31)) |
```

```
 87                 *(int *) &x = (0x3f800000 & ((125 - k) >> 31)) |
118                     (0x80000000 & hx);
119                 return (x);
120         } else {
 90         } else
121                 /* signal invalid if x is a SNaN */
122                 return (x * ONEF);      /* +0 -> *1 for Cheetah */
123         }
124 #endif /* ! codereview */
125 }

127 int
128 irintf(float x)
129 {
 93 irintf(float x) {
130         float v;
131         int hx, k;

133         hx = *(int *)&x;
 97         hx = *(int *) &x;
134         k = (hx & ~0x80000000) >> 23;
135         v = xf[((k - 150) >> 31) & (1 - (hx >> 31))];
136         return ((int)((float)(x + v) - v));
100         return ((int) ((float) (x + v) - v));
137 }

139 int
140 nintf(float x)
141 {
104 nintf(float x) {
142         int hx, ix, k, j, m;
143         volatile float dummy __unused;

145         hx = *(int *)&x;
108         hx = *(int *) &x;
146         k = (hx & ~0x80000000) >> 23;

148 #endif /* ! codereview */
149         if (((k - 126) ^ (k - 150)) < 0) {
150                 ix = (hx & 0x00ffffff) | 0x800000;
151                 m = 149 - k;
152                 j = 1 << m;

154 #endif /* ! codereview */
155                 if ((ix & (j + j - 1)) != 0)
156                         dummy = HUGEF + x;

158 #endif /* ! codereview */
159                 hx = hx >> 31;
160                 return ((((ix + j) >> (m + 1)) ^ hx) - hx);
161         } else {
162                 return ((int)x);
163         }
110         } else
111                 return ((int) x);
164 }

166 float
167 rintf(float x)
168 {
115 rintf(float x) {
169         float w, v;
170         int hx, k;

172         hx = *(int *)&x;
119         hx = *(int *) &x;
```



```
173         k = (hx & ~0x80000000) >> 23;
174 #if defined(FPADD_TRAPS_INCOMPLETE_ON_NAN)
175         if (k >= 150)
176                 return (x * ONEF);

178 #endif /* ! codereview */
179         v = xf[1 - (hx >> 31)];
180 #else
181         v = xf[((k - 150) >> 31) & (1 - (hx >> 31))];
182 #endif
183         w = (float)(x + v);

124         w = (float) (x + v);
185         if (k < 127 && w == v)
186                 return (ZEROF * x);
187         else
188                 return (w - v);
189 }
```

_____*unchanged_portion_omitted_*

```
**********************************************************
    1731 Thu Jun  6 20:39:22 2019
new/usr/src/lib/libm/common/R/scalbf.c
11210 libm should be cstyle(10NBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

#endif /* ! codereview */
/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2005 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __scalbf = scalbf

#include "libm.h"

float
scalbf(float x, float y)
{
scalbf(float x, float y) {
        int ix, iy, hx, hy, n;

        ix = *(int *)&x;
        iy = *(int *)&y;
        hx = ix & ~0x80000000;
        hy = iy & ~0x80000000;

        if (hx > 0x7f800000 || hy >= 0x7f800000) {
                /* x is nan or y is inf or nan */
                return ((iy < 0) ? x / -y : x *y);
                return ((iy < 0)? x / -y : x * y);
        }

        /* see if y is an integer without raising inexact */
        if (hy >= 0x4b000000) {
                /* |y| >= 2^23, so it must be an integer */
                n = (iy < 0) ? -65000 : 65000;
                n = (iy < 0)? -65000 : 65000;
        } else if (hy < 0x3f800000) {
                /* |y| < 1, so it must be zero or non-integer */
                return ((hy == 0) ? x : (x - x) / (x - x));
```



```
                return ((hy == 0)? x : (x - x) / (x - x));
        } else {
                if (hy & ((1 << (0x96 - (hy >> 23))) - 1))
                        return ((y - y) / (y - y));

#endif /* ! codereview */
                n = (int)y;
        }

#endif /* ! codereview */
        return (scalbnf(x, n));
}
```

```
**********************************************************
    2402 Thu Jun  6 20:39:22 2019
new/usr/src/lib/libm/common/R/scalbnf.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __scalbnf = scalbnf

#include "libm.h"
#include <float.h>                      /* FLT_MAX, FLT_MIN */
#include <stdlib.h>                     /* abs */

static const float twom25f = 2.98023223876953125e-8F;

#endif /* ! codereview */
#if defined(__x86)
static const float two23f = 8388608.0F;
#else
/*
 * v: a non-zero subnormal |x|; returns [-22, 0]
 */
static int
ilogbf_biased(unsigned v)
{
ilogbf_biased(unsigned v) {
        int r = -22;

        if (v & 0xffff0000)
                r += 16, v >>= 16;

#endif /* ! codereview */
        if (v & 0xff00)
                r += 8, v >>= 8;

#endif /* ! codereview */
        if (v & 0xf0)
```



```
                r += 4, v >>= 4;

#endif /* ! codereview */
        v <<= 1;
        return (r + ((0xffffaa50 >> v) & 0x3));
}
#endif /* defined(__x86) */

float
scalbnf(float x, int n)
{
        int *px = (int *)&x, ix, k;
scalbnf(float x, int n) {
        int *px = (int *) &x, ix, k;

        ix = *px & ~0x80000000;
        k = ix >> 23;

#endif /* ! codereview */
        if (k == 0xff)
#if defined(FPADD_TRAPS_INCOMPLETE_ON_NAN)
                return (ix > 0x7f800000 ? x * x : x);
#else
                return (x + x);
#endif

#endif /* ! codereview */
        if (ix == 0 || n == 0)
                return (x);

#endif /* ! codereview */
        if (k == 0) {
#if defined(__x86)
                x *= two23f;
                k = ((*px & ~0x80000000) >> 23) - 23;
#else
                k = ilogbf_biased(ix);
                *px = (*px & 0x80000000) | (ix << (-k + 1));
#endif
        }

        if ((unsigned)abs(n) >= 131072) /* cast to unsigned for -2^31 */
        if ((unsigned) abs(n) >= 131072)        /* cast to unsigned for -2^31 */
                n >>= 1;                /* avoid subsequent integer overflow */

#endif /* ! codereview */
        k += n;

#endif /* ! codereview */
        if (k > 0xfe)
                return (FLT_MAX * copysignf(FLT_MAX, x));

#endif /* ! codereview */
        if (k <= -25)
                return (FLT_MIN * copysignf(FLT_MIN, x));

#endif /* ! codereview */
        if (k > 0) {
                *px = (*px & ~0x7f800000) | (k << 23);
                return (x);
        }

#endif /* ! codereview */
        k += 25;
        *px = (*px & ~0x7f800000) | (k << 23);
        return (x * twom25f);
```

```
}
```

```
**********************************************************
    1071 Thu Jun  6 20:39:23 2019
new/usr/src/lib/libm/common/R/signgamf.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __signgamf = signgamf

#include "libm.h"

int signgamf = 0;
```

```
**********************************************************
    1337 Thu Jun  6 20:39:23 2019
new/usr/src/lib/libm/common/R/significandf.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __significandf = significandf

#include "libm.h"

float
significandf(float x)
{
        int ix = *(int *)&x & ~0x80000000;
significandf(float x) {
        int ix = *(int *) &x & ~0x80000000;

        if (ix == 0 || ix >= 0x7f800000)        /* 0/+-Inf/NaN */
#if defined(FPADD_TRAPS_INCOMPLETE_ON_NAN)
                return (ix > 0x7f800000 ? x * x : x);
#else
                return (x + x);
#endif
        else
                return (scalbnf(x, -ilogbf(x)));
}
______unchanged_portion_omitted_
```

```
**********************************************************
    5067 Thu Jun  6 20:39:24 2019
new/usr/src/lib/libm/common/R/sincosf.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
   1 /*
   2  * CDDL HEADER START
   3  *
   4  * The contents of this file are subject to the terms of the
   5  * Common Development and Distribution License (the "License").
   6  * You may not use this file except in compliance with the License.
   7  *
   8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
   9  * or http://www.opensolaris.org/os/licensing.
  10  * See the License for the specific language governing permissions
  11  * and limitations under the License.
  12  *
  13  * When distributing Covered Code, include this CDDL HEADER in each
  14  * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  15  * If applicable, add the following below this CDDL HEADER, with the
  16  * fields enclosed by brackets "[]" replaced with your own identifying
  17  * information: Portions Copyright [yyyy] [name of copyright owner]
  18  *
  19  * CDDL HEADER END
  20  */

  22 #endif /* ! codereview */
  23 /*
  24  * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
  25  */

  27 #endif /* ! codereview */
  28 /*
  29  * Copyright 2005 Sun Microsystems, Inc.  All rights reserved.
  30  * Use is subject to license terms.
  31  */

  33 #pragma weak __sincosf = sincosf


  21 /* INDENT OFF */
  36 /*
  37  * For |x| < pi/4, let z = x * x, and approximate sin(x) by
  38  *
  39  *      S(x) = x(S0 + S1*z)(S2 + S3*z + z*z)
  40  * where
  41  *      S0 =   1.85735322054308378716204874632872525989806770558e-0003,
  42  *      S1 =  -1.95035094218403635082921458859320791358115801259e-0004,
  43  *      S2 =   5.38400550766074785970952495168558701485841707252e+0002,
  44  *      S3 =  -3.31975110777873728964197739157371509422022905947e+0001,
  45  *
  46  * with error bounded by |(sin(x) - S(x))/x| < 2**(-28.2), and
  47  * cos(x) by
  48  *
  49  *      C(x) = (C0 + C1*z + C2*z*z) * (C3 + C4*z + z*z)
  50  * where
  51  *      C0 =   1.09349482127188401868272000389539985058873853699e-0003
  52  *      C1 =  -5.03324285989649793980347000549202268661076750 91e-0004
  53  *      C2 =   2.43792880266971107750418061559602239831538067410e-0005
  54  *      C3 =   9.14499072605666582228127405245558035523741471271e+0002
  55  *      C4 =  -3.63151270591815439197122504991683846785293207730e+0001
  56  *
  57  * with error bounded by |cos(x) - C(x)| < 2**(-34.2).
  58  */
  45 /* INDENT ON */
```



```
  60 #include "libm.h"

  62 extern const int _TBL_ipio2_inf[];
  63 extern int __rem_pio2m(double *, double *, int, int, int, const int *);

  65 #endif /* ! codereview */
  66 #if defined(__i386) && !defined(__amd64)
  67 extern int __swapRP(int);
  68 #endif

  70 static const double C[] = {
  71         1.85735322054308378716204874632872525989806770558e-0003,
  72         -1.95035094218403635082921458859320791358115801259e-0004,
  73         5.38400550766074785970952495168558701485841707252e+0002,
  74         -3.31975110777873728964197739157371509422022905947e+0001,
  75         1.09349482127188401868272000389539985058873853699e-0003,
  76         -5.03324285989649793980347000549202268661076750 91e-0004,
  77         2.43792880266971107750418061559602239831538067410e-0005,
  78         9.14499072605666582228127405245558035523741471271e+0002,
  79         -3.63151270591815439197122504991683846785293207730e+0001,
  80         0.636619772367581343075535,     /* 2^ -1  * 1.45F306DC9C883 */
  81         0.5,
  82         1.57079632673412561416 6,       /* 2^  0  * 1.921FB54400000 */
  83         6.07710050650619260147 5e-11,   /* 2^-34  * 1.0B4611A626331 */
  84 };

  86 #define S0              C[0]
  87 #define S1              C[1]
  88 #define S2              C[2]
  89 #define S3              C[3]
  90 #define C0              C[4]
  91 #define C1              C[5]
  92 #define C2              C[6]
  93 #define C3              C[7]
  94 #define C4              C[8]
  95 #define invpio2         C[9]
  96 #define half            C[10]
  97 #define pio2_1          C[11]
  98 #define pio2_t          C[12]

 100 void
 101 sincosf(float x, float *s, float *c)
 102 {
 103         double y, z, w;
 104         float f, g;
 105         int n, ix, hx, hy;
 106         volatile int i __unused;

 108         hx = *((int *)&x);
 109         ix = hx & 0x7fffffff;

 111         y = (double)x;

 113         if (ix <= 0x4016cbe4) {                 /* |x| < 3*pi/4 */
 114                 if (ix <= 0x3f490fdb) {         /* |x| < pi/4 */
 115                         if (ix <= 0x39800000) { /* |x| <= 2**-12 */
 116                                 i = (int)y;
 117 #ifdef lint
 118                                 i = i;
 119 #endif
 120                                 *s = x;
 121                                 *c = 1.0f;
 122                                 return;
 123                         }

 125 #endif /* ! codereview */
```

```
126                         z = y * y;
127                         *s = (float)((y * (S0 + z * S1)) * (S2 + z * (S3 + z)));
128                         *c = (float)(((C0 + z * C1) + (z * z) * C2) * (C3 + z *
129                             (C4 + z)));
 51                         *s = (float)((y * (S0 + z * S1)) *
 52                             (S2 + z * (S3 + z)));
 53                         *c = (float)(((C0 + z * C1) + (z * z) * C2) *
 54                             (C3 + z * (C4 + z)));
130                 } else if (hx > 0) {
131                         y = (y - pio2_1) - pio2_t;
132                         z = y * y;
133                         *s = (float)(((C0 + z * C1) + (z * z) * C2) * (C3 + z *
134                             (C4 + z)));
135                         *c = (float)-((y * (S0 + z * S1)) * (S2 + z * (S3 +
136                             z)));
 58                         *s = (float)(((C0 + z * C1) + (z * z) * C2) *
 59                             (C3 + z * (C4 + z)));
 60                         *c = (float)-((y * (S0 + z * S1)) *
 61                             (S2 + z * (S3 + z)));
137                 } else {
138                         y = (y + pio2_1) + pio2_t;
139                         z = y * y;
140                         *s = (float)-(((C0 + z * C1) + (z * z) * C2) * (C3 + z *
141                             (C4 + z)));
142                         *c = (float)((y * (S0 + z * S1)) * (S2 + z * (S3 + z)));
 65                         *s = (float)-(((C0 + z * C1) + (z * z) * C2) *
 66                             (C3 + z * (C4 + z)));
 67                         *c = (float)((y * (S0 + z * S1)) *
 68                             (S2 + z * (S3 + z)));
143                 }

145 #endif /* ! codereview */
146                 return;
147         } else if (ix <= 0x49c90fdb) {  /* |x| < 2^19*pi */
148 #if defined(__i386) && !defined(__amd64)
149                 int rp;

151                 rp = __swapRP(fp_extended);
152 #endif
153                 w = y * invpio2;

155 #endif /* ! codereview */
156                 if (hx < 0)
157                         n = (int)(w - half);
158                 else
159                         n = (int)(w + half);

161 #endif /* ! codereview */
162                 y = (y - n * pio2_1) - n * pio2_t;
163 #if defined(__i386) && !defined(__amd64)
164                 if (rp != fp_extended)
165                         (void) __swapRP(rp);
166 #endif
167         } else {
168                 if (ix >= 0x7f800000) {
169                         *s = *c = x / x;
170                         return;
171                 }

173 #endif /* ! codereview */
174                 hy = ((int *)&y)[HIWORD];
175                 n = ((hy >> 20) & 0x7ff) - 1046;
176                 ((int *)&w)[HIWORD] = (hy & 0xfffff) | 0x41600000;
177                 ((int *)&w)[LOWORD] = ((int *)&y)[LOWORD];
178                 n = __rem_pio2m(&w, &y, n, 1, 0, _TBL_ipio2_inf);
```



```
180 #endif /* ! codereview */
181                 if (hy < 0) {
182                         y = -y;
183                         n = -n;
184                 }
185         }

187         z = y * y;
188         f = (float)((y * (S0 + z * S1)) * (S2 + z * (S3 + z)));
189         g = (float)(((C0 + z * C1) + (z * z) * C2) * (C3 + z * (C4 + z)));

 70         g = (float)(((C0 + z * C1) + (z * z) * C2) *
 71             (C3 + z * (C4 + z)));
191         if (n & 2) {
192                 f = -f;
193                 g = -g;
194         }

196 #endif /* ! codereview */
197         if (n & 1) {
198                 *s = g;
199                 *c = -f;
200         } else {
201                 *s = f;
202                 *c = g;
203         }
204 }
```

```
**********************************************************
    1348 Thu Jun  6 20:39:24 2019
new/usr/src/lib/libm/common/R/sincospif.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
   1 /*
   2  * CDDL HEADER START
   3  *
   4  * The contents of this file are subject to the terms of the
   5  * Common Development and Distribution License (the "License").
   6  * You may not use this file except in compliance with the License.
   7  *
   8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
   9  * or http://www.opensolaris.org/os/licensing.
  10  * See the License for the specific language governing permissions
  11  * and limitations under the License.
  12  *
  13  * When distributing Covered Code, include this CDDL HEADER in each
  14  * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  15  * If applicable, add the following below this CDDL HEADER, with the
  16  * fields enclosed by brackets "[]" replaced with your own identifying
  17  * information: Portions Copyright [yyyy] [name of copyright owner]
  18  *
  19  * CDDL HEADER END
  20  */

  22 /*
  23  * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
  24  */

  26 #endif /* ! codereview */
  27 /*
  28  * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
  29  * Use is subject to license terms.
  30  */

  32 #include "libm.h"

  34 extern void sincospi(double, double *, double *);

  36 void
  37 sincospif(float x, float *s, float *c)
  38 {
  25 sincospif(float x, float *s, float *c) {
  39         double ds, dc;

  41 #if defined(FPADD_TRAPS_INCOMPLETE_ON_NAN)
  42         if (isnanf(x)) {
  29         if (isnanf(x))
  43                 *s = *c = x * x;
  44         } else {
  31         else {
  45 #endif
  46         sincospi((double)x, &ds, &dc);
  47         *s = (float)ds;
  48         *c = (float)dc;
  33         sincospi((double) x, &ds, &dc);
  34         *s = (float) ds;
  35         *c = (float) dc;
  49 #if defined(FPADD_TRAPS_INCOMPLETE_ON_NAN)
  50 }
  37         }
  51 #endif
  52 }
______unchanged_portion_omitted_
```

```
**********************************************************
    3943 Thu Jun  6 20:39:25 2019
new/usr/src/lib/libm/common/R/sinf.c
11210 libm should be cstyle(10NBLD) clean
**********************************************************
   1 /*
   2  * CDDL HEADER START
   3  *
   4  * The contents of this file are subject to the terms of the
   5  * Common Development and Distribution License (the "License").
   6  * You may not use this file except in compliance with the License.
   7  *
   8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
   9  * or http://www.opensolaris.org/os/licensing.
  10  * See the License for the specific language governing permissions
  11  * and limitations under the License.
  12  *
  13  * When distributing Covered Code, include this CDDL HEADER in each
  14  * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  15  * If applicable, add the following below this CDDL HEADER, with the
  16  * fields enclosed by brackets "[]" replaced with your own identifying
  17  * information: Portions Copyright [yyyy] [name of copyright owner]
  18  *
  19  * CDDL HEADER END
  20  */

  22 #endif /* ! codereview */
  23 /*
  24  * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
  25  */

  27 #endif /* ! codereview */
  28 /*
  29  * Copyright 2005 Sun Microsystems, Inc.  All rights reserved.
  30  * Use is subject to license terms.
  31  */

  33 #pragma weak __sinf = sinf

  35 /*
  36  * See sincosf.c
  37  */

  39 #include "libm.h"

  41 extern const int _TBL_ipio2_inf[];
  42 extern int __rem_pio2m(double *, double *, int, int, int, const int *);

  44 #endif /* ! codereview */
  45 #if defined(__i386) && !defined(__amd64)
  46 extern int __swapRP(int);
  47 #endif

  49 static const double C[] = {
  50         1.857353220543083787162048746328725259898067705 58e-0003,
  51         -1.950350942184036350829214588593207913581158012 59e-0004,
  52         5.384005507660747859709524951685587014858417072 52e+0002,
  53         -3.319751107778737289641977391573715094220229059 47e+0001,
  54         1.093494821271884018682720003895399850588738536 99e-0003,
  55         -5.033242859899649793980347000549202268661076750 91e-0004,
  56         2.437928802669711077504180615596022398315380674 10e-0005,
  57         9.144990726056665822281274052455580355237414712 71e+0002,
  58         -3.631512705918154391971225049916838467852932077 30e+0001,
  59         0.636619772367581343075535,     /* 2^ -1  * 1.45F306DC9C883 */
  60         0.5,
  61         1.570796326734125614166,        /* 2^  0  * 1.921FB54400000 */
```



```
  62         6.077100506506192601475e-11,    /* 2^-34  * 1.0B4611A626331 */
  63 };

  65 #define S0              C[0]
  66 #define S1              C[1]
  67 #define S2              C[2]
  68 #define S3              C[3]
  69 #define C0              C[4]
  70 #define C1              C[5]
  71 #define C2              C[6]
  72 #define C3              C[7]
  73 #define C4              C[8]
  74 #define invpio2         C[9]
  75 #define half            C[10]
  76 #define pio2_1          C[11]
  77 #define pio2_t          C[12]

  79 float
  80 sinf(float x)
  81 {
  82         double y, z, w;
  83         float f;
  84         int n, ix, hx, hy;
  85         volatile int i __unused;

  87         hx = *((int *)&x);
  88         ix = hx & 0x7fffffff;

  90         y = (double)x;

  92         if (ix <= 0x4016cbe4) {                 /* |x| < 3*pi/4 */
  93                 if (ix <= 0x3f490fdb) {         /* |x| < pi/4 */
  94                         if (ix <= 0x39800000) { /* |x| <= 2**-12 */
  95                                 i = (int)y;
  96 #ifdef lint
  97                                 i = i;
  98 #endif
  99                                 return (x);
 100                         }

 102 #endif /* ! codereview */
 103                         z = y * y;
 104                         return ((float)((y * (S0 + z * S1)) * (S2 + z *
 105                             (S3 + z))));
  21                         return ((float)((y * (S0 + z * S1)) *
  22                             (S2 + z * (S3 + z))));
 106                 } else if (hx > 0) {
 107                         y = (y - pio2_1) - pio2_t;
 108                         z = y * y;
 109                         return ((float)(((C0 + z * C1) + (z * z) * C2) * (C3 +
 110                             z * (C4 + z))));
  26                         return ((float)(((C0 + z * C1) + (z * z) * C2) *
  27                             (C3 + z * (C4 + z))));
 111                 } else {
 112                         y = (y + pio2_1) + pio2_t;
 113                         z = y * y;
 114                         return ((float)-(((C0 + z * C1) + (z * z) * C2) * (C3 +
 115                             z * (C4 + z))));
  31                         return ((float)-(((C0 + z * C1) + (z * z) * C2) *
  32                             (C3 + z * (C4 + z))));
 116                 }
 117         } else if (ix <= 0x49c90fdb) {  /* |x| < 2^19*pi */
 118 #if defined(__i386) && !defined(__amd64)
 119                 int rp;

 121                 rp = __swapRP(fp_extended);
```

```
#endif
                w = y * invpio2;

#endif /* ! codereview */
                if (hx < 0)
                        n = (int)(w - half);
                else
                        n = (int)(w + half);

#endif /* ! codereview */
                y = (y - n * pio2_1) - n * pio2_t;
#if defined(__i386) && !defined(__amd64)
                if (rp != fp_extended)
                        (void) __swapRP(rp);
#endif
        } else {
                if (ix >= 0x7f800000)
                        return (x / x);         /* sin(Inf or NaN) is NaN */

#endif /* ! codereview */
                hy = ((int *)&y)[HIWORD];
                n = ((hy >> 20) & 0x7ff) - 1046;
                ((int *)&w)[HIWORD] = (hy & 0xfffff) | 0x41600000;
                ((int *)&w)[LOWORD] = ((int *)&y)[LOWORD];
                n = __rem_pio2m(&w, &y, n, 1, 0, _TBL_ipio2_inf);

#endif /* ! codereview */
                if (hy < 0) {
                        y = -y;
                        n = -n;
                }
        }

        if (n & 1) {
                /* compute cos y */
                z = y * y;
                f = (float)(((C0 + z * C1) + (z * z) * C2) * (C3 + z * (C4 +
                    z)));
                f = (float)(((C0 + z * C1) + (z * z) * C2) *
                    (C3 + z * (C4 + z)));
        } else {
                /* compute sin y */
                z = y * y;
                f = (float)((y * (S0 + z * S1)) * (S2 + z * (S3 + z)));
        }

        return ((n & 2) ? -f : f);
        return ((n & 2)? -f : f);
}
```

_____*unchanged_portion_omitted_*

```
**********************************************************
    1397 Thu Jun  6 20:39:25 2019
new/usr/src/lib/libm/common/R/sinhf.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

#endif /* ! codereview */
/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2005 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __sinhf = sinhf

#include "libm.h"

float
sinhf(float x)
{
sinhf(float x) {
        double s;
        float w;
        int hx, ix;

        hx = *(int *)&x;
        ix = hx & ~0x80000000;

#endif /* ! codereview */
        if (ix >= 0x7f800000) {
                /* sinhf(x) is x if x is +-Inf or NaN */
                return (x * 1.0f);
        }

#endif /* ! codereview */
        if (ix >= 0x43000000)           /* sinhf(x) trivially overflows */
                s = (hx < 0) ? -1.0e100 : 1.0e100;
                s = (hx < 0)? -1.0e100 : 1.0e100;
        else
                s = sinh((double)x);

#endif /* ! codereview */
```



```
        w = (float)s;
        return (w);
}
```

```
**********************************************************
    1138 Thu Jun  6 20:39:26 2019
new/usr/src/lib/libm/common/R/sqrtf.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

#endif /* ! codereview */
/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2005 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __sqrtf = sqrtf

#include "libm.h"


extern float __inline_sqrtf(float);

float
sqrtf(float x)
{
sqrtf(float x) {
        return (__inline_sqrtf(x));
}
```

```
**********************************************************
    4353 Thu Jun  6 20:39:26 2019
new/usr/src/lib/libm/common/R/tanf.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

#endif /* ! codereview */
/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2005 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __tanf = tanf

#include "libm.h"

extern const int _TBL_ipio2_inf[];
extern int __rem_pio2m(double *, double *, int, int, int, const int *);

#endif /* ! codereview */
#if defined(__i386) && !defined(__amd64)
extern int __swapRP(int);
#endif

static const double C[] = {
        1.0,
        4.46066928428959230679140546271810308098793029785e-0003,
        4.92165316309189027066395283327437937259674072266e+0000,
        -7.11410648161473480044492134766187518835067749023e-0001,
        4.08549808374053391446523164631798863410949707031e+0000,
        2.50411070398050927821032018982805311679840087891e+0000,
        1.11492064560251158411574579076841473579406738281e+0001,
        -1.50565540968422650891511693771462887525558471680e+0000,
        -1.81484378783492950500431106775067746639251708898e+0000,
        3.33333599753283564129740961178251089664 1e-0001,
        2.99999759824836376154166828200686722993 9e+00,
        0.63661977236758134307553 5,     /* 2^ -1  * 1.45F306DC9C883 */
        0.5,
        1.57079632673412561416 6,        /* 2^  0  * 1.921FB54400000 */
        6.07710050650619260147 5e-11,    /* 2^-34  * 1.0B4611A626331 */
};
```



```
#define one             C[0]
#define P0              C[1]
#define P1              C[2]
#define P2              C[3]
#define P3              C[4]
#define P4              C[5]
#define P5              C[6]
#define P6              C[7]
#define P7              C[8]
#define T0              C[9]
#define T1              C[10]
#define invpio2         C[11]
#define half            C[12]
#define pio2_1          C[13]
#define pio2_t          C[14]

float
tanf(float x)
{
        double y, z, w;
        float f;
        int n, ix, hx, hy;
        volatile int i __unused;

        hx = *((int *)&x);
        ix = hx & 0x7fffffff;

        y = (double)x;

        if (ix <= 0x4016cbe4) {                         /* |x| < 3*pi/4 */
                if (ix <= 0x3f490fdb) {                 /* |x| < pi/4 */
                        if (ix < 0x3c000000) {          /* |x| < 2**-7 */
                                if (ix <= 0x39800000) { /* |x| < 2**-12 */
                                        i = (int)y;
#ifdef lint
                                        i = i;
#endif
                                        return (x);
                                }

#endif /* ! codereview */
                                return ((float)((y * T0) * (T1 + y * y)));
                        }

#endif /* ! codereview */
                        z = y * y;
                        return ((float)(((P0 * y) * (P1 + z * (P2 + z)) * (P3 +
                            z * (P4 + z))) * (P5 + z * (P6 + z *
                            (P7 + z)))));
                        return ((float)(((P0 * y) * (P1 + z * (P2 + z)) *
                            (P3 + z * (P4 + z))) *
                            (P5 + z * (P6 + z * (P7 + z)))));
                }

#endif /* ! codereview */
                if (hx > 0)
                        y = (y - pio2_1) - pio2_t;
                else
                        y = (y + pio2_1) + pio2_t;

#endif /* ! codereview */
                hy = ((int *)&y)[HIWORD] & ~0x80000000;

#endif /* ! codereview */
                if (hy < 0x3f800000) {  /* |y| < 2**-7 */
```

```
                        z = (y * T0) * (T1 + y * y);
                        return ((float)(-one / z));
                }

#endif /* ! codereview */
                z = y * y;
                w = ((P0 * y) * (P1 + z * (P2 + z)) * (P3 + z * (P4 + z))) *
                    (P5 + z * (P6 + z * (P7 + z)));
                return ((float)(-one / w));
        }

        if (ix <= 0x49c90fdb) {         /* |x| < 2^19*pi */
#if defined(__i386) && !defined(__amd64)
                int rp;

                rp = __swapRP(fp_extended);
#endif
                w = y * invpio2;

#endif /* ! codereview */
                if (hx < 0)
                        n = (int)(w - half);
                else
                        n = (int)(w + half);

#endif /* ! codereview */
                y = (y - n * pio2_1) - n * pio2_t;
#if defined(__i386) && !defined(__amd64)
                if (rp != fp_extended)
                        (void) __swapRP(rp);
#endif
        } else {
                if (ix >= 0x7f800000)
                        return (x / x);         /* sin(Inf or NaN) is NaN */

#endif /* ! codereview */
                hy = ((int *)&y)[HIWORD];
                n = ((hy >> 20) & 0x7ff) - 1046;
                ((int *)&w)[HIWORD] = (hy & 0xfffff) | 0x41600000;
                ((int *)&w)[LOWORD] = ((int *)&y)[LOWORD];
                n = __rem_pio2m(&w, &y, n, 1, 0, _TBL_ipio2_inf);

#endif /* ! codereview */
                if (hy < 0) {
                        y = -y;
                        n = -n;
                }
        }

        hy = ((int *)&y)[HIWORD] & ~0x80000000;

#endif /* ! codereview */
        if (hy < 0x3f800000) {          /* |y| < 2**-7 */
                z = (y * T0) * (T1 + y * y);
                f = ((n & 1) == 0) ? (float)z : (float)(-one / z);
                f = ((n & 1) == 0)? (float)z : (float)(-one / z);
                return (f);
        }

#endif /* ! codereview */
        z = y * y;
        w = ((P0 * y) * (P1 + z * (P2 + z)) * (P3 + z * (P4 + z))) * (P5 + z *
            (P6 + z * (P7 + z)));
        f = ((n & 1) == 0) ? (float)w : (float)(-one / w);
        w = ((P0 * y) * (P1 + z * (P2 + z)) * (P3 + z * (P4 + z))) *
            (P5 + z * (P6 + z * (P7 + z)));
```



```
        f = ((n & 1) == 0)? (float)w : (float)(-one / w);
        return (f);
}
```

_____*unchanged_portion_omitted_*

```
**********************************************************
    1196 Thu Jun  6 20:39:27 2019
new/usr/src/lib/libm/common/R/tanhf.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
   1 /*
   2  * CDDL HEADER START
   3  *
   4  * The contents of this file are subject to the terms of the
   5  * Common Development and Distribution License (the "License").
   6  * You may not use this file except in compliance with the License.
   7  *
   8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
   9  * or http://www.opensolaris.org/os/licensing.
  10  * See the License for the specific language governing permissions
  11  * and limitations under the License.
  12  *
  13  * When distributing Covered Code, include this CDDL HEADER in each
  14  * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  15  * If applicable, add the following below this CDDL HEADER, with the
  16  * fields enclosed by brackets "[]" replaced with your own identifying
  17  * information: Portions Copyright [yyyy] [name of copyright owner]
  18  *
  19  * CDDL HEADER END
  20  */

  22 /*
  23  * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
  24  */

  26 #endif /* ! codereview */
  27 /*
  28  * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
  29  * Use is subject to license terms.
  30  */

  32 #pragma weak __tanhf = tanhf

  34 #include "libm.h"

  36 float
  37 tanhf(float x)
  38 {
  25 tanhf(float x) {
  39 #if defined(FPADD_TRAPS_INCOMPLETE_ON_NAN)
  40         if (isnanf(x))
  41                 return (x * x);
  42         else
  43 #endif
  44         return ((float)tanh((double)x));
  31         return ((float) tanh((double) x));
  45 }
______unchanged_portion_omitted_
```

```
**********************************************************
    4664 Thu Jun  6 20:39:27 2019
new/usr/src/lib/libm/common/complex/cabs.c
11210 libm should be cstyle(10NBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

#endif /* ! codereview */
/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2005 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __cabs = cabs

#include <math.h>
#include "complex_wrapper.h"

/*
 * If C were the only standard we cared about, cabs could just call
 * hypot.  Unfortunately, various other standards say that hypot must
 * call matherr and/or set errno to ERANGE when the result overflows.
 * Since cabs should do neither of these things, we have to either
 * make hypot a wrapper on another internal function or duplicate
 * the hypot implementation here.  I've chosen to do the latter.
 */

static const double zero = 0.0,
static const double
        zero = 0.0,
        onep1u = 1.0000000000000022204e+00,    /* 0x3ff00000 1 = 1+2**-52 */
        twom53 = 1.1102230246251565404e-16,    /* 0x3ca00000 0 = 2**-53 */
        twom768 = 6.441148769597133308e-232,   /* 2^-768 */
        two768 = 1.552518092300708935e+231;    /* 2^768 */

double
cabs(dcomplex z)
{
        double x, y, xh, yh, w, ax, ay;
        int i, j, nx, ny, ix, iy, iscale = 0;
        unsigned lx, ly;
```



```
        x = D_RE(z);
        y = D_IM(z);

        ix = ((int *)&x)[HIWORD] & ~0x80000000;
        lx = ((int *)&x)[LOWORD];
        iy = ((int *)&y)[HIWORD] & ~0x80000000;
        ly = ((int *)&y)[LOWORD];

        /* force ax = |x| ~>~ ay = |y| */
        if (iy > ix) {
                ax = fabs(y);
                ay = fabs(x);
                i = ix;
                ix = iy;
                iy = i;
                i = lx;
                lx = ly;
                ly = i;
        } else {
                ax = fabs(x);
                ay = fabs(y);
        }

#endif /* ! codereview */
        nx = ix >> 20;
        ny = iy >> 20;
        j = nx - ny;

        if (nx >= 0x5f3) {
                /* x >= 2^500 (x*x or y*y may overflow) */
                if (nx == 0x7ff) {
                        /* inf or NaN, signal of sNaN */
                        if (((ix - 0x7ff00000) | lx) == 0)
                                return ((ax == ay) ? ay : ax);
                                return ((ax == ay)? ay : ax);
                        else if (((iy - 0x7ff00000) | ly) == 0)
                                return ((ay == ax) ? ax : ay);
                                return ((ay == ax)? ax : ay);
                        else
                                return (ax * ay);
                } else if (j > 32) {
                        /* x >> y */
                        if (j <= 53)
                                ay *= twom53;

#endif /* ! codereview */
                        ax += ay;
                        return (ax);
                }

#endif /* ! codereview */
                ax *= twom768;
                ay *= twom768;
                iscale = 2;
                ix -= 768 << 20;
                iy -= 768 << 20;
        } else if (ny < 0x23d) {
                /* y < 2^-450 (x*x or y*y may underflow) */
                if ((ix | lx) == 0)
                        return (ax);

#endif /* ! codereview */
                if ((iy | ly) == 0)
                        return (ax);

#endif /* ! codereview */
```

```
                if (j > 53)             /* x >> y */
                        return (ax + ay);

#endif /* ! codereview */
                iscale = 1;
                ax *= two768;
                ay *= two768;

#endif /* ! codereview */
                if (nx == 0) {
                        if (ax == zero) /* guard subnormal flush to zero */
                                return (ax);

#endif /* ! codereview */
                        ix = ((int *)&ax)[HIWORD];
                } else {
                        ix += 768 << 20;
                }

#endif /* ! codereview */
                if (ny == 0) {
                        if (ay == zero) /* guard subnormal flush to zero */
                                return (ax * twom768);

#endif /* ! codereview */
                        iy = ((int *)&ay)[HIWORD];
                } else {
                        iy += 768 << 20;
                }

#endif /* ! codereview */
                j = (ix >> 20) - (iy >> 20);

#endif /* ! codereview */
                if (j > 32) {
                        /* x >> y */
                        if (j <= 53)
                                ay *= twom53;

#endif /* ! codereview */
                        return ((ax + ay) * twom768);
                }
        } else if (j > 32) {
                /* x >> y */
                if (j <= 53)
                        ay *= twom53;

#endif /* ! codereview */
                return (ax + ay);
        }

        /*
         * Medium range ax and ay with max{|ax/ay|,|ay/ax|} bounded by 2^32.
         * First check rounding mode by comparing onep1u*onep1u with onep1u
         * + twom53.  Make sure the computation is done at run-time.
         */
        if (((lx | ly) << 5) == 0) {
                ay = ay * ay;
                ax += ay / (ax + sqrt(ax * ax + ay));
        } else if (onep1u * onep1u != onep1u + twom53) {
                /*
                 * round-to-zero, positive, negative mode
                 * magic formula with less than an ulp error
                 */
                /* round-to-zero, positive, negative mode */
                /* magic formula with less than an ulp error */
```



```
                w = sqrt(ax * ax + ay * ay);
                ax += ay / ((ax + w) / ay);
        } else {
                /* round-to-nearest mode */
                w = ax - ay;

#endif /* ! codereview */
                if (w > ay) {
                        ((int *)&xh)[HIWORD] = ix;
                        ((int *)&xh)[LOWORD] = 0;
                        ay = ay * ay + (ax - xh) * (ax + xh);
                        ax = sqrt(xh * xh + ay);
                } else {
                        ax = ax + ax;
                        ((int *)&xh)[HIWORD] = ix + 0x00100000;
                        ((int *)&xh)[LOWORD] = 0;
                        ((int *)&yh)[HIWORD] = iy;
                        ((int *)&yh)[LOWORD] = 0;
                        ay = w * w + ((ax - xh) * yh + (ay - yh) * ax);
                        ax = sqrt(xh * yh + ay);
                }
        }

#endif /* ! codereview */
        if (iscale > 0) {
                if (iscale == 1)
                        ax *= twom768;
                else
                        ax *= two768;   /* must generate side effect here */
        }

#endif /* ! codereview */
        return (ax);
}
```

```
**********************************************************
    1140 Thu Jun  6 20:39:28 2019
new/usr/src/lib/libm/common/complex/cabsf.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __cabsf = cabsf

#include "libm.h"
#include "complex_wrapper.h"

float
cabsf(fcomplex z)
{
cabsf(fcomplex z) {
        return (hypotf(F_RE(z), F_IM(z)));
}
______unchanged_portion_omitted_
```

```
**********************************************************
    1149 Thu Jun  6 20:39:28 2019
new/usr/src/lib/libm/common/complex/cabsl.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __cabsl = cabsl

#include "libm.h"
#include "complex_wrapper.h"

long double
cabsl(ldcomplex z)
{
cabsl(ldcomplex z) {
        return (hypotl(LD_RE(z), LD_IM(z)));
}
______unchanged_portion_omitted_
```

```
**********************************************************
   12644 Thu Jun  6 20:39:29 2019
new/usr/src/lib/libm/common/complex/cacos.c
11210 libm should be cstyle(10NBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __cacos = cacos


/* INDENT OFF */
/*
 * dcomplex cacos(dcomplex z);
 *
 * Alogrithm
 * (based on T.E.Hull, Thomas F. Fairgrieve and Ping Tak Peter Tang's
 * paper "Implementing the Complex Arcsine and Arccosine Functins Using
 * Exception Handling", ACM TOMS, Vol 23, pp 299-335)
 *
 * The principal value of complex inverse cosine function cacos(z),
 * where z = x+iy, can be defined by
 *
 *      cacos(z) = acos(B) - i sign(y) log (A + sqrt(A*A-1)),
 *
 * where the log function is the natural log, and
 *                                  _____________       _____________
 *       1    /     2    2      1    /     2    2
 *  A = ---  / (x+1)  + y   +  ---  / (x-1)  + y
 *       2 \/                   2 \/
 *                                  _____________       _____________
 *       1    /     2    2      1    /     2    2
 *  B = ---  / (x+1)  + y   -  ---  / (x-1)  + y   .
 *       2 \/                   2 \/
 *
 * The Branch cuts are on the real line from -inf to -1 and from 1 to inf.
 * The real and imaginary parts are based on Abramowitz and Stegun
 * [Handbook of Mathematic Functions, 1972].  The sign of the imaginary
```



```
 * part is chosen to be the generally considered the principal value of
 * this function.
 *
 * Notes:1. A is the average of the distances from z to the points (1,0)
 *          and (-1,0) in the complex z-plane, and in particular A>=1.
 *       2. B is in [-1,1], and A*B = x
 *
 * Basic relations
 *    cacos(conj(z)) = conj(cacos(z))
 *    cacos(-z)      = pi   - cacos(z)
 *    cacos( z)      = pi/2 - casin(z)
 *
 * Special cases (conform to ISO/IEC 9899:1999(E)):
 *    cacos(+-0  + i y  ) = pi/2 - i y for y is +-0, +-inf, NaN
 *    cacos( x   + i inf) = pi/2 - i inf for all x
 *    cacos( x   + i NaN) = NaN  + i NaN with invalid for non-zero finite x
 *    cacos(-inf + i y  ) = pi   - i inf for finite +y
 *    cacos( inf + i y  ) = 0    - i inf for finite +y
 *    cacos(-inf + i inf) = 3pi/4- i inf
 *    cacos( inf + i inf) = pi/4 - i inf
 *    cacos(+-inf+ i NaN) = NaN  - i inf (sign of imaginary is unspecified)
 *    cacos(NaN  + i y  ) = NaN  + i NaN with invalid for finite y
 *    cacos(NaN  + i inf) = NaN  - i inf
 *    cacos(NaN  + i NaN) = NaN  + i NaN
 *
 * Special Regions (better formula for accuracy and for avoiding spurious
 * overflow or underflow) (all x and y are assumed nonnegative):
 *  case 1: y = 0
 *  case 2: tiny y relative to x-1: y <= ulp(0.5)*|x-1|
 *  case 3: tiny y: y < 4 sqrt(u), where u = minimum normal number
 *  case 4: huge y relative to x+1: y >= (1+x)/ulp(0.5)
 *  case 5: huge x and y: x and y >= sqrt(M)/8, where M = maximum normal number
 *  case 6: tiny x: x < 4 sqrt(u)
 *  --------
 *  case        1 & 2. y=0 or y/|x-1| is tiny. We have
 *                  _____________             _____________
 *              /       2    2             /       y    2
 *            / (x+1)  + y   =  |x+-1| / 1 + (------)
 *          \/                            \/       |x+-1|
 *
 *                                                 1     y    2
 *                            ~  |x+-1| ( 1 + --- (------)  )
 *                                                 2   |x+-1|
 *
 *                                              2
 *                                             y
 *                            = |x+-1| + --------.
 *                                        2|x+-1|
 *
 *      Consequently, it is not difficult to see that
 *                                    2
 *                                   y
 *                     [ 1 + ------------ ,     if x < 1,
 *                     [      2(1+x)(1-x)
 *                     [
 *                     [
 *                     [ x,                     if x = 1 (y = 0),
 *                     [
 *              A ~=   [            2
 *                     [         x * y
 *                     [ x + ------------ ~ x,  if x > 1
 *                     [      2(x+1)(x-1)
 *
 *      and hence
 *                          ______                                    2
 *                        / 2                   y                  y
```

```
 *               A + \/ A  - 1  ~  1 + ---------------- + -----------, if x < 1,
 *                                     sqrt((x+1)(1-x))   2(x+1)(1-x)
 *
 *                                ~  x + sqrt((x-1)*(x+1)),             if x >= 1.
 *
 *                                          2
 *                                         y
 *                          [ x(1 - -----------) ~ x,  if x < 1,
 *                          [       2(1+x)(1-x)
 *               B = x/A  ~ [
 *                          [ 1,                       if x = 1,
 *                          [
 *                          [           2
 *                          [          y
 *                          [ 1 - ------------ ,       if x > 1,
 *                          [      2(x+1)(x-1)
 *      Thus
 *                            [ acos(x) - i y/sqrt((x-1)*(x+1)),      if x < 1,
 *                            [
 *               cacos(x+i*y)~ [ 0 - i 0,                             if x = 1,
 *                            [
 *                            [ y/sqrt(x*x-1) - i log(x+sqrt(x*x-1)), if x > 1.
 *
 *      Note: y/sqrt(x*x-1) ~ y/x when x >= 2**26.
 *  case 3. y < 4 sqrt(u), where u = minimum normal x.
 *      After case 1 and 2, this will only occurs when x=1. When x=1, we have
 *         A = (sqrt(4+y*y)+y)/2 ~ 1 + y/2 + y^2/8 + ...
 *      and
 *         B = 1/A = 1 - y/2 + y^2/8 + ...
 *      Since
 *         cos(sqrt(y)) ~ 1 - y/2 + ...
 *      we have, for the real part,
 *         acos(B) ~ acos(1 - y/2) ~ sqrt(y)
 *      For the imaginary part,
 *         log(A+sqrt(A*A-1)) ~ log(1+y/2+sqrt(2*y/2))
 *                            = log(1+y/2+sqrt(y))
 *                            = (y/2+sqrt(y)) - (y/2+sqrt(y))^2/2 + ...
 *                            ~ sqrt(y) - y*(sqrt(y)+y/2)/2
 *                            ~ sqrt(y)
 *
 *  case 4. y >= (x+1)/ulp(0.5). In this case, A ~ y and B ~ x/y. Thus
 *         real part = acos(B) ~ pi/2
 *      and
 *         imag part = log(y+sqrt(y*y-one))
 *
 *  case 5. Both x and y are large: x and y > sqrt(M)/8, where M = maximum x
 *      In this case,
 *         A ~ sqrt(x*x+y*y)
 *         B ~ x/sqrt(x*x+y*y).
 *      Thus
 *         real part = acos(B) = atan(y/x),
 *         imag part = log(A+sqrt(A*A-1)) ~ log(2A)
 *                   = log(2) + 0.5*log(x*x+y*y)
 *                   = log(2) + log(y) + 0.5*log(1+(x/y)^2)
 *
 *  case 6. x < 4 sqrt(u). In this case, we have
 *          A ~ sqrt(1+y*y), B = x/sqrt(1+y*y).
 *      Since B is tiny, we have
 *          real part = acos(B) ~ pi/2
 *          imag part = log(A+sqrt(A*A-1)) = log (A+sqrt(y*y))
 *                    = log(y+sqrt(1+y*y))
 *                    = 0.5*log(y^2+2ysqrt(1+y^2)+1+y^2)
 *                    = 0.5*log(1+2y(y+sqrt(1+y^2)));
 *                    = 0.5*log1p(2y(y+A));
 *
```



```
 *      cacos(z) = acos(B) - i sign(y) log (A + sqrt(A*A-1)),
 */
/* INDENT ON */

#include "libm.h"
#include "complex_wrapper.h"

static const double zero = 0.0,
/* INDENT OFF */
static const double
        zero = 0.0,
        one = 1.0,
        E = 1.1102230246251565404 2e-16, /* 2**-53 */
        ln2 = 6.93147180559945286227e-01,
        pi = 3.14159265358979311599796346 85,
        pi_l = 1.22464679914735317 7e-16,
        pi_2 = 1.57079632679489655 8e+00,
        pi_2_l = 6.123233995736765886e-17,
        pi_4 = 0.78539816339744827899949,
        pi_4_l = 3.06161699786838294 3e-17,
        pi3_4 = 2.35619449019234483 6998,
        pi3_4_l = 9.18485099360514882 9195e-17,
        Foursqrtu = 5.96667258496016539463e-154,        /* 2**(-509) */
        Acrossover = 1.5,
        Bcrossover = 0.6417,
        half = 0.5;

/* INDENT ON */

dcomplex
cacos(dcomplex z)
{
cacos(dcomplex z) {
        double x, y, t, R, S, A, Am1, B, y2, xm1, xp1, Apx;
        int ix, iy, hx, hy;
        unsigned lx, ly;
        dcomplex ans;

        x = D_RE(z);
        y = D_IM(z);
        hx = HI_WORD(x);
        lx = LO_WORD(x);
        hy = HI_WORD(y);
        ly = LO_WORD(y);
        ix = hx & 0x7fffffff;
        iy = hy & 0x7fffffff;

        /* x is 0 */
        if ((ix | lx) == 0) {
                if (((iy | ly) == 0) || (iy >= 0x7ff00000)) {
                        D_RE(ans) = pi_2;
                        D_IM(ans) = -y;
                        return (ans);
                }
        }

        /* |y| is inf or NaN */
        if (iy >= 0x7ff00000) {
                if (ISINF(iy, ly)) {    /* cacos(x + i inf) = pi/2  - i inf */
                        D_IM(ans) = -y;

#endif /* ! codereview */
                        if (ix < 0x7ff00000) {
                                D_RE(ans) = pi_2 + pi_2_l;
                        } else if (ISINF(ix, lx)) {
                                if (hx >= 0)
```

```
                                        D_RE(ans) = pi_4 + pi_4_l;
                                else
                                        D_RE(ans) = pi3_4 + pi3_4_l;
                        } else {
                                D_RE(ans) = x;
                        }
                } else {                /* cacos(x + i NaN) = NaN  + i NaN */
                        D_RE(ans) = y + x;

#endif /* ! codereview */
                        if (ISINF(ix, lx))
                                D_IM(ans) = -fabs(x);
                        else
                                D_IM(ans) = y;
                }

#endif /* ! codereview */
                return (ans);
        }

        x = fabs(x);
        y = fabs(y);

        /* x is inf or NaN */
        if (ix >= 0x7ff00000) {         /* x is inf or NaN */
                if (ISINF(ix, lx)) {    /* x is INF */
                        D_IM(ans) = -x;

#endif /* ! codereview */
                        if (iy >= 0x7ff00000) {
                                if (ISINF(iy, ly)) {
                                        /*
                                         * cacos(inf + i inf) = pi/4 - i inf
                                         * cacos(-inf+ i inf) =3pi/4 - i inf
                                         */
                                        /* INDENT OFF */
                                        /* cacos(inf + i inf) = pi/4 - i inf */
                                        /* cacos(-inf+ i inf) =3pi/4 - i inf */
                                        /* INDENT ON */
                                        if (hx >= 0)
                                                D_RE(ans) = pi_4 + pi_4_l;
                                        else
                                                D_RE(ans) = pi3_4 + pi3_4_l;
                                } else {
                                        /*
                                         * cacos(inf + i NaN) = NaN  - i inf
                                         */
                                } else
                                        /* INDENT OFF */
                                        /* cacos(inf + i NaN) = NaN  - i inf  */
                                        /* INDENT ON */
                                        D_RE(ans) = y + y;
                                }
#endif /* ! codereview */
                        } else
                        /*
                         * cacos(inf + iy ) = 0  - i inf
                         * cacos(-inf+ iy  ) = pi - i inf
                         */
                        if (hx >= 0) {
                                /* INDENT OFF */
                                /* cacos(inf + iy ) = 0  - i inf */
                                /* cacos(-inf+ iy  ) = pi - i inf */
                                /* INDENT ON */
                        if (hx >= 0)
                                D_RE(ans) = zero;
```



```
                        } else {
                        else
                                D_RE(ans) = pi + pi_l;
                        }
#endif /* ! codereview */
                } else {                /* x is NaN */

                        /* INDENT OFF */
                        /*
                         * cacos(NaN + i inf) = NaN  - i inf
                         * cacos(NaN + i y  ) = NaN  + i NaN
                         * cacos(NaN + i NaN) = NaN  + i NaN
                         */
                        /* INDENT ON */
                        D_RE(ans) = x + y;

                        if (iy >= 0x7ff00000)
                        if (iy >= 0x7ff00000) {
                                D_IM(ans) = -y;
                        else
                        } else {
                                D_IM(ans) = x;
                }

                }
                if (hy < 0)
                        D_IM(ans) = -D_IM(ans);

#endif /* ! codereview */
                return (ans);
        }

        if ((iy | ly) == 0) {           /* region 1: y=0 */
                if (ix < 0x3ff00000) {  /* |x| < 1 */
                        D_RE(ans) = acos(x);
                        D_IM(ans) = zero;
                } else {
                        D_RE(ans) = zero;

                        if (ix >= 0x43500000) {         /* |x| >= 2**54 */
                        if (ix >= 0x43500000)   /* |x| >= 2**54 */
                                D_IM(ans) = ln2 + log(x);
                        } else if (ix >= 0x3ff80000) {  /* x > Acrossover */
                        else if (ix >= 0x3ff80000)      /* x > Acrossover */
                                D_IM(ans) = log(x + sqrt((x - one) * (x +
                                    one)));
                        } else {
                        else {
                                xm1 = x - one;
                                D_IM(ans) = log1p(xm1 + sqrt(xm1 * (x + one)));
                        }
                }
        } else if (y <= E * fabs(x - one)) {    /* region 2: y < tiny*|x-1| */
                if (ix < 0x3ff00000) {          /* x < 1 */
                        D_RE(ans) = acos(x);
                        D_IM(ans) = y / sqrt((one + x) * (one - x));
                } else if (ix >= 0x43500000) {  /* |x| >= 2**54 */
                        D_RE(ans) = y / x;
                        D_IM(ans) = ln2 + log(x);
                } else {
                        t = sqrt((x - one) * (x + one));
                        D_RE(ans) = y / t;

#endif /* ! codereview */
                        if (ix >= 0x3ff80000)   /* x > Acrossover */
                                D_IM(ans) = log(x + t);
```

```
                        else
                                D_IM(ans) = log1p((x - one) + t);
                }
        } else if (y < Foursqrtu) {     /* region 3 */
                t = sqrt(y);
                D_RE(ans) = t;
                D_IM(ans) = t;
        } else if (E * y - one >= x) {                          /* region 4 */
                D_RE(ans) = pi_2;
                D_IM(ans) = ln2 + log(y);
        } else if (ix >= 0x5fc00000 || iy >= 0x5fc00000) {      /* x,y>2**509 */
                /* region 5: x+1 or y is very large (>= sqrt(max)/8) */
                t = x / y;
                D_RE(ans) = atan(y / x);
                D_IM(ans) = ln2 + log(y) + half * log1p(t * t);
        } else if (x < Foursqrtu) {
                /* region 6: x is very small, < 4sqrt(min) */
                D_RE(ans) = pi_2;
                A = sqrt(one + y * y);

#endif /* ! codereview */
                if (iy >= 0x3ff80000)   /* if y > Acrossover */
                        D_IM(ans) = log(y + A);
                else
                        D_IM(ans) = half * log1p((y + y) * (y + A));
        } else {                        /* safe region */
                y2 = y * y;
                xp1 = x + one;
                xm1 = x - one;
                R = sqrt(xp1 * xp1 + y2);
                S = sqrt(xm1 * xm1 + y2);
                A = half * (R + S);
                B = x / A;

                if (B <= Bcrossover) {
                if (B <= Bcrossover)
                        D_RE(ans) = acos(B);
                } else {        /* use atan and an accurate approx to a-x */
                else {          /* use atan and an accurate approx to a-x */
                        Apx = A + x;

#endif /* ! codereview */
                        if (x <= one)
                                D_RE(ans) = atan(sqrt(half * Apx * (y2 / (R +
                                    xp1) + (S - xm1))) / x);
                        else
                                D_RE(ans) = atan((y * sqrt(half * (Apx / (R +
                                    xp1) + Apx / (S + xm1)))) / x);
                }

#endif /* ! codereview */
                if (A <= Acrossover) {
                        /* use log1p and an accurate approx to A-1 */
                        if (x < one)
                                Am1 = half * (y2 / (R + xp1) + y2 / (S - xm1));
                        else
                                Am1 = half * (y2 / (R + xp1) + (S + xm1));

#endif /* ! codereview */
                        D_IM(ans) = log1p(Am1 + sqrt(Am1 * (A + one)));
                } else {
                        D_IM(ans) = log(A + sqrt(A * A - one));
                }
        }

#endif /* ! codereview */
```



```
        if (hx < 0)
                D_RE(ans) = pi - D_RE(ans);

#endif /* ! codereview */
        if (hy >= 0)
                D_IM(ans) = -D_IM(ans);

#endif /* ! codereview */
        return (ans);
}
```

```
**********************************************************
    1310 Thu Jun  6 20:39:29 2019
new/usr/src/lib/libm/common/complex/cacosf.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
   1 /*
   2  * CDDL HEADER START
   3  *
   4  * The contents of this file are subject to the terms of the
   5  * Common Development and Distribution License (the "License").
   6  * You may not use this file except in compliance with the License.
   7  *
   8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
   9  * or http://www.opensolaris.org/os/licensing.
  10  * See the License for the specific language governing permissions
  11  * and limitations under the License.
  12  *
  13  * When distributing Covered Code, include this CDDL HEADER in each
  14  * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  15  * If applicable, add the following below this CDDL HEADER, with the
  16  * fields enclosed by brackets "[]" replaced with your own identifying
  17  * information: Portions Copyright [yyyy] [name of copyright owner]
  18  *
  19  * CDDL HEADER END
  20  */

  22 /*
  23  * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
  24  */

  26 #endif /* ! codereview */
  27 /*
  28  * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
  29  * Use is subject to license terms.
  30  */

  32 #pragma weak __cacosf = cacosf

  34 #include "libm.h"
  35 #include "complex_wrapper.h"

  37 fcomplex
  38 cacosf(fcomplex z)
  39 {
  25 cacosf(fcomplex z) {
  40         dcomplex dz, dans;
  41         fcomplex ans;

  43         D_RE(dz) = (double)(F_RE(z));
  44         D_IM(dz) = (double)(F_IM(z));
  29         D_RE(dz) = (double) (F_RE(z));
  30         D_IM(dz) = (double) (F_IM(z));
  45         dans = cacos(dz);
  46         F_RE(ans) = (float)(D_RE(dans));
  47         F_IM(ans) = (float)(D_IM(dans));
  32         F_RE(ans) = (float) (D_RE(dans));
  33         F_IM(ans) = (float) (D_IM(dans));
  48         return (ans);
  49 }
______unchanged_portion_omitted_
```

```
**********************************************************
    1709 Thu Jun  6 20:39:30 2019
new/usr/src/lib/libm/common/complex/cacosh.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __cacosh = cacosh


/* INDENT OFF */
/*
 * dcomplex cacosh(dcomplex z);
 *      cacosh z = +-i cacos z .
 * In order to make conj(cacosh(z))=cacosh(conj(z)),
 * we define
 *      cacosh z = sign(Im(z))*i cacos z .
 *
 */
/* INDENT ON */

#include "libm.h"                       /* fabs/isnan/isinf/signbit */
#include "complex_wrapper.h"

/* need to work on special cases according to spec */

dcomplex
cacosh(dcomplex z)
{
cacosh(dcomplex z) {
        dcomplex w, ans;
        double x, y;

        w = cacos(z);
        x = D_RE(z);
        y = D_IM(z);
```



```
#endif /* ! codereview */
        if (isnan(y)) {
                D_IM(ans) = y + y;

#endif /* ! codereview */
                if (isinf(x))
                        D_RE(ans) = fabs(x);
                else
                        D_RE(ans) = y;
        } else if (signbit(y) == 0) {
                D_RE(ans) = -D_IM(w);
                D_IM(ans) = D_RE(w);
        } else {
                D_RE(ans) = D_IM(w);
                D_IM(ans) = -D_RE(w);
        }

#endif /* ! codereview */
        return (ans);
}
```

```
**********************************************************
    1369 Thu Jun  6 20:39:30 2019
new/usr/src/lib/libm/common/complex/cacoshf.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __cacoshf = cacoshf

#include "libm.h"
#include "complex_wrapper.h"

/* need to work on special cases according to spec */

fcomplex
cacoshf(fcomplex z)
{
cacoshf(fcomplex z) {
        dcomplex dz, dans;
        fcomplex ans;

        D_RE(dz) = (double)(F_RE(z));
        D_IM(dz) = (double)(F_IM(z));
        D_RE(dz) = (double) (F_RE(z));
        D_IM(dz) = (double) (F_IM(z));
        dans = cacosh(dz);
        F_RE(ans) = (float)(D_RE(dans));
        F_IM(ans) = (float)(D_IM(dans));
        F_RE(ans) = (float) (D_RE(dans));
        F_IM(ans) = (float) (D_IM(dans));
        return (ans);
}
______unchanged_portion_omitted_
```

```
**********************************************************
    1714 Thu Jun  6 20:39:31 2019
new/usr/src/lib/libm/common/complex/cacoshl.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __cacoshl = cacoshl

#include "libm.h"                       /* fabsl/isnanl/isinfl/signbitl */
#include "complex_wrapper.h"
#include "longdouble.h"


/* INDENT OFF */
/*
 * ldcomplex cacoshl(ldcomplex z);
 *      cacosh z = +-i cacos z .
 * In order to make conj(cacosh(z))=cacosh(conj(z)),
 * we define
 *      cacosh z = sign(Im(z))*i cacos z .
 *
 */
/* INDENT ON */

ldcomplex
cacoshl(ldcomplex z)
{
cacoshl(ldcomplex z) {
        ldcomplex w, ans;
        long double x, y;

        w = cacosl(z);
        x = LD_RE(z);
        y = LD_IM(z);

#endif /* ! codereview */
```



```
        if (isnanl(y)) {
                LD_IM(ans) = y + y;

#endif /* ! codereview */
                if (isinfl(x))
                        LD_RE(ans) = fabsl(x);
                else
                        LD_RE(ans) = y;
        } else if (signbitl(y) == 0) {
                LD_RE(ans) = -LD_IM(w);
                LD_IM(ans) = LD_RE(w);
        } else {
                LD_RE(ans) = LD_IM(w);
                LD_IM(ans) = -LD_RE(w);
        }

#endif /* ! codereview */
        return (ans);
}
```

```
**********************************************************
    7626 Thu Jun  6 20:39:32 2019
new/usr/src/lib/libm/common/complex/cacosl.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
   1 /*
   2  * CDDL HEADER START
   3  *
   4  * The contents of this file are subject to the terms of the
   5  * Common Development and Distribution License (the "License").
   6  * You may not use this file except in compliance with the License.
   7  *
   8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
   9  * or http://www.opensolaris.org/os/licensing.
  10  * See the License for the specific language governing permissions
  11  * and limitations under the License.
  12  *
  13  * When distributing Covered Code, include this CDDL HEADER in each
  14  * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  15  * If applicable, add the following below this CDDL HEADER, with the
  16  * fields enclosed by brackets "[]" replaced with your own identifying
  17  * information: Portions Copyright [yyyy] [name of copyright owner]
  18  *
  19  * CDDL HEADER END
  20  */

  22 /*
  23  * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
  24  */

  26 #endif /* ! codereview */
  27 /*
  28  * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
  29  * Use is subject to license terms.
  30  */

  32 #pragma weak __cacosl = cacosl

  34 #include "libm.h"       /* acosl/atanl/fabsl/isinfl/log1pl/logl/sqrtl */
  35 #include "complex_wrapper.h"
  36 #include "longdouble.h"

  38 /* BEGIN CSTYLED */
  39 static const long double zero = 0.0L,
  40         one = 1.0L,
  41         Acrossover = 1.5L,
  42         Bcrossover = 0.6417L,
  43         half = 0.5L,
  44         ln2 = 6.931471805599453094172321214581765680755e-0001L,
  45         Foursqrtu = 7.334415470219388662485649568193932663825 5e-2466L,  /* 2**-8
  25 /* INDENT OFF */
  26 static const long double
  27 zero = 0.0L,
  28 one = 1.0L,
  29 Acrossover = 1.5L,
  30 Bcrossover = 0.6417L,
  31 half = 0.5L,
  32 ln2 = 6.931471805599453094172321214581765680755e-0001L,
  33 Foursqrtu = 7.334415470219388662485649568193932663825 5e-2466L,  /* 2**-8189 */
  46 #if defined(__x86)
  47         E = 5.421010862427522170037264004349708557 1289e-20L,    /* 2**-64 */
  48         pi = 3.141592653589793238295968524909085317631252110004425048828125L,
  49         pi_l = 1.666748583704175665659172893706807721468195923078e-19L,
  50         pi_2 = 1.570796326794896619147984262454542658815 6260L,
  51         pi_2_l = 8.333742918520878328295864468534038607340 9796e-20L,
  52         pi_4 = 0.785398163397448309573992131227271329407813027501106262207 03125L
```



```
  53         pi_4_l = 4.166871459260439164147932234267019303670489807695410e-20L,
  54         pi3_4 = 2.356194490192344928721976393681813988223439082503318786621093 75
  55         pi3_4_l = 1.250061437778131749244379670280105791101146942308e-19L;
  35 E = 5.421010862427522170037264004349708557 1289e-20L,            /* 2**-64 */
  36 pi = 3.141592653589793238295968524909085317631252110004425048828125L,
  37 pi_l = 1.666748583704175665659172893706807721468195923078e-19L,
  38 pi_2 = 1.570796326794896619147984262454542658815 6260L,
  39 pi_2_l = 8.333742918520878328295864468534038607340 9796e-20L,
  40 pi_4 = 0.785398163397448309573992131227271329407813027501106262207 03125L,
  41 pi_4_l = 4.166871459260439164147932234267019303670489807695410e-20L,
  42 pi3_4 = 2.356194490192344928721976393681813988223439082503318786621093 75L,
  43 pi3_4_l = 1.250061437778131749244379670280105791101146942308e-19L;
  56 #else
  57         E = 9.629649721936179265279889712924636592690 5e-35L,    /* 2**-113 */
  58         pi = 3.141592653589793238462643383279502797479068098137295573004504 3318L
  59         pi_l = 8.671810130123781024797044026043351968762323346256530341775935686
  60         pi_2 = 1.570796326794896619231321691639751398739 5340L,
  61         pi_2_l = 4.335905065061890512398522013021675984381 1616e-35L,
  62         pi_4 = 0.785398163397448309615660845819875699369767024534323893251126L,
  63         pi_4_l = 2.167952532530945256199261006510837992190580836564132585443e-35
  64         pi3_4 = 2.356194490192344928846982537459627098109301073602971679753378 24
  65         pi3_4_l = 6.503857597592835768597783019532513976571742509692397756331e-3
  45 E = 9.629649721936179265279889712924636592690 5e-35L,            /* 2**-113 */
  46 pi = 3.141592653589793238462643383279502797479068098137295573004504 3318L,
  47 pi_l = 8.671810130123781024797044026043351968762323346256530341775935686 2e-35L,
  48 pi_2 = 1.570796326794896619231321691639751398739 5340L,
  49 pi_2_l = 4.335905065061890512398522013021675984381 1616e-35L,
  50 pi_4 = 0.785398163397448309615660845819875699369767024534323893251126L,
  51 pi_4_l = 2.167952532530945256199261006510837992190580836564132585443e-35L,
  52 pi3_4 = 2.356194490192344928846982537459627098109301073602971679753378 24L,
  53 pi3_4_l = 6.503857597592835768597783019532513976571742509692397756331e-35L;
  66 #endif
  67 /* END CSTYLED */
  55 /* INDENT ON */

  69 #if defined(__x86)
  70 static const int ip1 = 0x40400000;      /* 2**65 */
  71 #else
  72 static const int ip1 = 0x40710000;      /* 2**114 */
  73 #endif

  75 ldcomplex
  76 cacosl(ldcomplex z)
  77 {
  64 cacosl(ldcomplex z) {
  78         long double x, y, t, R, S, A, Am1, B, y2, xm1, xp1, Apx;
  79         int ix, iy, hx, hy;
  80         ldcomplex ans;

  82         x = LD_RE(z);
  83         y = LD_IM(z);
  84         hx = HI_XWORD(x);
  85         hy = HI_XWORD(y);
  86         ix = hx & 0x7fffffff;
  87         iy = hy & 0x7fffffff;

  89         /* x is 0 */
  90         if (x == zero) {
  91                 if (y == zero || (iy >= 0x7fff0000)) {
  92                         LD_RE(ans) = pi_2 + pi_2_l;
  93                         LD_IM(ans) = -y;
  94                         return (ans);
  95                 }
  96         }

  98         /* |y| is inf or NaN */
```

```
 99         if (iy >= 0x7fff0000) {
100                 if (isinfl(y)) {        /* cacos(x + i inf) =  pi/2 - i inf */
101                         LD_IM(ans) = -y;

103 #endif /* ! codereview */
104                         if (ix < 0x7fff0000) {
105                                 LD_RE(ans) = pi_2 + pi_2_l;
106                         } else if (isinfl(x)) {
107                                 if (hx >= 0)
108                                         LD_RE(ans) = pi_4 + pi_4_l;
109                                 else
110                                         LD_RE(ans) = pi3_4 + pi3_4_l;
111                         } else {
112                                 LD_RE(ans) = x;
113                         }
114                 } else {                /* cacos(x + i NaN) = NaN  + i NaN */
115                         LD_RE(ans) = y + x;

117 #endif /* ! codereview */
118                         if (isinfl(x))
119                                 LD_IM(ans) = -fabsl(x);
120                         else
121                                 LD_IM(ans) = y;
122                 }

124 #endif /* ! codereview */
125                 return (ans);
126         }

128         y = fabsl(y);

130         if (ix >= 0x7fff0000) {         /* x is inf or NaN */
131                 if (isinfl(x)) {        /* x is INF */
132                         LD_IM(ans) = -fabsl(x);

134 #endif /* ! codereview */
135                         if (iy >= 0x7fff0000) {
136                                 if (isinfl(y)) {
137                                         /*
138                                          * cacos(inf + i inf) = pi/4 - i inf
139                                          * cacos(-inf+ i inf) =3pi/4 - i inf
140                                          */
 89                                         /* INDENT OFF */
 90                                         /* cacos(inf + i inf) = pi/4 - i inf */
 91                                         /* cacos(-inf+ i inf) =3pi/4 - i inf */
 92                                         /* INDENT ON */
141                                         if (hx >= 0)
142                                                 LD_RE(ans) = pi_4 + pi_4_l;
143                                         else
144                                                 LD_RE(ans) = pi3_4 + pi3_4_l;
145                                 } else {
146                                         /*
147                                          * cacos(inf + i NaN) = NaN  - i inf
148                                          */
 97                                 } else
 98                                         /* INDENT OFF */
 99                                         /* cacos(inf + i NaN) = NaN  - i inf  */
100                                         /* INDENT ON */
149                                         LD_RE(ans) = y + y;
150                                 }
151 #endif /* ! codereview */
152                         } else {
153                                 /*
154                                  * cacos(inf + iy ) = 0  - i inf
155                                  * cacos(-inf+ iy  ) = pi - i inf
156                                  */
```



```
102                                 /* INDENT OFF */
103                                 /* cacos(inf + iy ) = 0  - i inf */
104                                 /* cacos(-inf+ iy  ) = pi - i inf */
105                                 /* INDENT ON */
157                                 if (hx >= 0)
158                                         LD_RE(ans) = zero;
159                                 else
160                                         LD_RE(ans) = pi + pi_l;
161                         }
162                 } else {                /* x is NaN */

112                         /* INDENT OFF */
164                         /*
165                          * cacos(NaN + i inf) = NaN  - i inf
166                          * cacos(NaN + i y  ) = NaN  + i NaN
167                          * cacos(NaN + i NaN) = NaN  + i NaN
168                          */
118                         /* INDENT ON */
169                         LD_RE(ans) = x + y;

171                         if (iy >= 0x7fff0000)
120                         if (iy >= 0x7fff0000) {
172                                 LD_IM(ans) = -y;
173                         else
122                         } else {
174                                 LD_IM(ans) = x;
175                 }

125                 }
177                 if (hy < 0)
178                         LD_IM(ans) = -LD_IM(ans);

180 #endif /* ! codereview */
181                 return (ans);
182         }

184         if (y == zero) {                /* region 1: y=0 */
185                 if (ix < 0x3fff0000) {  /* |x| < 1 */
186                         LD_RE(ans) = acosl(x);
187                         LD_IM(ans) = zero;
188                 } else {
189                         LD_RE(ans) = zero;
190                         x = fabsl(x);

192                         if (ix >= ip1) { /* i386 ? 2**65 : 2**114 */
128                         if (ix >= ip1)  /* i386 ? 2**65 : 2**114 */
193                                 LD_IM(ans) = ln2 + logl(x);
194                         } else if (ix >= 0x3fff8000) {  /* x > Acrossover */
130                         else if (ix >= 0x3fff8000)      /* x > Acrossover */
195                                 LD_IM(ans) = logl(x + sqrtl((x - one) * (x +
196                                     one)));
197                         } else {
133                         else {
198                                 xm1 = x - one;
199                                 LD_IM(ans) = log1pl(xm1 + sqrtl(xm1 * (x +
200                                     one)));
201                         }
202                 }
203         } else if (y <= E * fabsl(fabsl(x) - one)) {
204                 /* region 2: y < tiny*||x|-1| */
205                 if (ix < 0x3fff0000) {  /* x < 1 */
206                         LD_RE(ans) = acosl(x);
207                         x = fabsl(x);
208                         LD_IM(ans) = y / sqrtl((one + x) * (one - x));
209                 } else if (ix >= ip1) { /* i386 ? 2**65 : 2**114 */
210                         if (hx >= 0) {
```

```
                        if (hx >= 0)
                                LD_RE(ans) = y / x;
                        } else {
                                if (ix >= ip1 + 0x00040000) {
                        else {
                                if (ix >= ip1 + 0x00040000)
                                        LD_RE(ans) = pi + pi_l;
                                } else {
                                else {
                                        t = pi_l + y / x;
                                        LD_RE(ans) = pi + t;
                                }
                        }

#endif /* ! codereview */
                        LD_IM(ans) = ln2 + logl(fabsl(x));
                } else {
                        x = fabsl(x);
                        t = sqrtl((x - one) * (x + one));
                        LD_RE(ans) = (hx >= 0) ? y / t : pi - (y / t - pi_l);

                        LD_RE(ans) = (hx >= 0)? y / t : pi - (y / t - pi_l);
                        if (ix >= 0x3fff8000)   /* x > Acrossover */
                                LD_IM(ans) = logl(x + t);
                        else
                                LD_IM(ans) = log1pl(t - (one - x));
                }
        } else if (y < Foursqrtu) {     /* region 3 */
                t = sqrtl(y);
                LD_RE(ans) = (hx >= 0) ? t : pi + pi_l;
                LD_RE(ans) = (hx >= 0)? t : pi + pi_l;
                LD_IM(ans) = t;
        } else if (E * y - one >= fabsl(x)) {   /* region 4 */
                LD_RE(ans) = pi_2 + pi_2_l;
                LD_IM(ans) = ln2 + logl(y);
        } else if (ix >= 0x5ffb0000 || iy >= 0x5ffb0000) {
                /* region 5: x+1 and y are both (>= sqrt(max)/8) i.e. 2**8188 */
                t = x / y;
                LD_RE(ans) = atan2l(y, x);
                LD_IM(ans) = ln2 + logl(y) + half * log1pl(t * t);
        } else if (fabsl(x) < Foursqrtu) {
                /* region 6: x is very small, < 4sqrt(min) */
                LD_RE(ans) = pi_2 + pi_2_l;
                A = sqrtl(one + y * y);

#endif /* ! codereview */
                if (iy >= 0x3fff8000)   /* if y > Acrossover */
                        LD_IM(ans) = logl(y + A);
                else
                        LD_IM(ans) = half * log1pl((y + y) * (y + A));
        } else {                        /* safe region */
                t = fabsl(x);
                y2 = y * y;
                xp1 = t + one;
                xm1 = t - one;
                R = sqrtl(xp1 * xp1 + y2);
                S = sqrtl(xm1 * xm1 + y2);
                A = half * (R + S);
                B = t / A;

                if (B <= Bcrossover) {
                        LD_RE(ans) = (hx >= 0) ? acosl(B) : acosl(-B);
                } else {        /* use atan and an accurate approx to a-x */
                if (B <= Bcrossover)
                        LD_RE(ans) = (hx >= 0)? acosl(B) : acosl(-B);
                else {          /* use atan and an accurate approx to a-x */
```



```
                        Apx = A + t;

#endif /* ! codereview */
                        if (t <= one)
                                LD_RE(ans) = atan2l(sqrtl(half * Apx * (y2 /
                                    (R + xp1) + (S - xm1))), x);
                        else
                                LD_RE(ans) = atan2l((y * sqrtl(half * (Apx /
                                    (R + xp1) + Apx / (S + xm1)))), x);
                }

#endif /* ! codereview */
                if (A <= Acrossover) {
                        /* use log1p and an accurate approx to A-1 */
                        if (ix < 0x3fff0000)
                                Am1 = half * (y2 / (R + xp1) + y2 / (S - xm1));
                        else
                                Am1 = half * (y2 / (R + xp1) + (S + xm1));

#endif /* ! codereview */
                        LD_IM(ans) = log1pl(Am1 + sqrtl(Am1 * (A + one)));
                } else {
                        LD_IM(ans) = logl(A + sqrtl(A * A - one));
                }
        }

        if (hy >= 0)
                LD_IM(ans) = -LD_IM(ans);

        return (ans);
}
```

```
**********************************************************
    1553 Thu Jun  6 20:39:32 2019
new/usr/src/lib/libm/common/complex/carg.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
   1 /*
   2  * CDDL HEADER START
   3  *
   4  * The contents of this file are subject to the terms of the
   5  * Common Development and Distribution License (the "License").
   6  * You may not use this file except in compliance with the License.
   7  *
   8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
   9  * or http://www.opensolaris.org/os/licensing.
  10  * See the License for the specific language governing permissions
  11  * and limitations under the License.
  12  *
  13  * When distributing Covered Code, include this CDDL HEADER in each
  14  * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  15  * If applicable, add the following below this CDDL HEADER, with the
  16  * fields enclosed by brackets "[]" replaced with your own identifying
  17  * information: Portions Copyright [yyyy] [name of copyright owner]
  18  *
  19  * CDDL HEADER END
  20  */

  22 #endif /* ! codereview */
  23 /*
  24  * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
  25  */

  27 #endif /* ! codereview */
  28 /*
  29  * Copyright 2005 Sun Microsystems, Inc.  All rights reserved.
  30  * Use is subject to license terms.
  31  */

  33 #pragma weak __carg = carg

  35 #include <math.h>                       /* atan2 */
  36 #include "complex_wrapper.h"

  38 static const double pi = 3.14159265358979311600e+00,
  21 static const double
  22         pi      = 3.14159265358979311600e+00,
  39         pi_lo = 1.22464679914735320717e-16;

  41 double
  42 carg(dcomplex z)
  43 {
  26 carg(dcomplex z) {
  44         int ix, iy;

  46         ix = ((int *)&(D_RE(z)))[HIWORD];
  47         iy = ((int *)&(D_IM(z)))[HIWORD];

  49 #endif /* ! codereview */
  50         if ((((ix | iy) & ~0x80000000) | ((int *)&(D_RE(z)))[LOWORD] |
  51             ((int *)&(D_IM(z)))[LOWORD]) == 0) {
  52                 /* x and y are both zero */
  53                 if (ix == 0)
  54                         return (D_IM(z));

  56                 return ((iy == 0) ? pi + pi_lo : -pi - pi_lo);
  31                 return ((iy == 0)? pi + pi_lo : -pi - pi_lo);
  57         }
```



```
  59 #endif /* ! codereview */
  60         return (atan2(D_IM(z), D_RE(z)));
  61 }
```

```
**********************************************************
    1155 Thu Jun  6 20:39:33 2019
new/usr/src/lib/libm/common/complex/cargf.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __cargf = cargf

#include "libm.h"                       /* atan2f */
#include "complex_wrapper.h"

float
cargf(fcomplex z)
{
cargf(fcomplex z) {
        return (atan2f(F_IM(z), F_RE(z)));
}
______unchanged_portion_omitted_
```

```
**********************************************************
    1149 Thu Jun  6 20:39:33 2019
new/usr/src/lib/libm/common/complex/cargl.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __cargl = cargl

#include "libm.h"
#include "complex_wrapper.h"

long double
cargl(ldcomplex z)
{
cargl(ldcomplex z) {
        return (atan2l(LD_IM(z), LD_RE(z)));
}
______unchanged_portion_omitted_
```

```
**********************************************************
   12177 Thu Jun  6 20:39:34 2019
new/usr/src/lib/libm/common/complex/casin.c
11210 libm should be cstyle(10NBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __casin = casin


/* INDENT OFF */
/*
 * dcomplex casin(dcomplex z);
 *
 * Alogrithm
 * (based on T.E.Hull, Thomas F. Fairgrieve and Ping Tak Peter Tang's
 * paper "Implementing the Complex Arcsine and Arccosine Functins Using
 * Exception Handling", ACM TOMS, Vol 23, pp 299-335)
 *
 * The principal value of complex inverse sine function casin(z),
 * where z = x+iy, can be defined by
 *
 *      casin(z) = asin(B) + i sign(y) log (A + sqrt(A*A-1)),
 *
 * where the log function is the natural log, and
 *                                  _____________       _____________
 *       1    /     2    2      1    /     2    2
 *  A = ---  / (x+1)  + y   +  ---  / (x-1)  + y
 *       2 \/                   2 \/
 *                                  _____________       _____________
 *       1    /     2    2      1    /     2    2
 *  B = ---  / (x+1)  + y   -  ---  / (x-1)  + y   .
 *       2 \/                   2 \/
 *
 * The Branch cuts are on the real line from -inf to -1 and from 1 to inf.
 * The real and imaginary parts are based on Abramowitz and Stegun
 * [Handbook of Mathematic Functions, 1972].  The sign of the imaginary
```



```
 * part is chosen to be the generally considered the principal value of
 * this function.
 *
 * Notes:1. A is the average of the distances from z to the points (1,0)
 *          and (-1,0) in the complex z-plane, and in particular A>=1.
 *       2. B is in [-1,1], and A*B = x.
 *
 * Special notes: if casin( x, y) = ( u, v), then
 *                 casin(-x, y) = (-u, v),
 *                 casin( x,-y) = ( u,-v),
 *    in general, we have casin(conj(z))     =  conj(casin(z))
 *                        casin(-z)          = -casin(z)
 *                        casin(z)           =  pi/2 - cacos(z)
 *
 * EXCEPTION CASES (conform to ISO/IEC 9899:1999(E)):
 *    casin( 0 + i 0   ) =  0    + i 0
 *    casin( 0 + i NaN ) =  0    + i NaN
 *    casin( x + i inf ) =  0    + i inf for finite x
 *    casin( x + i NaN ) =  NaN  + i NaN with invalid for finite x != 0
 *    casin(inf + iy   ) =  pi/2 + i inf finite y
 *    casin(inf + i inf) =  pi/4 + i inf
 *    casin(inf + i NaN) =  NaN  + i inf
 *    casin(NaN + i y  ) =  NaN  + i NaN for finite y
 *    casin(NaN + i inf) =  NaN  + i inf
 *    casin(NaN + i NaN) =  NaN  + i NaN
 *
 * Special Regions (better formula for accuracy and for avoiding spurious
 * overflow or underflow) (all x and y are assumed nonnegative):
 *  case 1: y = 0
 *  case 2: tiny y relative to x-1: y <= ulp(0.5)*|x-1|
 *  case 3: tiny y: y < 4 sqrt(u), where u = minimum normal number
 *  case 4: huge y relative to x+1: y >= (1+x)/ulp(0.5)
 *  case 5: huge x and y: x and y >= sqrt(M)/8, where M = maximum normal number
 *  case 6: tiny x: x < 4 sqrt(u)
 *  --------
 *  case        1 & 2. y=0 or y/|x-1| is tiny. We have
 *                  _____________                _____________
 *              /      2    2                /       y    2
 *             / (x+-1)  + y   =  |x+-1| / 1 + (------)
 *           \/                          \/        |x+-1|
 *
 *                                                  1      y   2
 *                             ~  |x+-1| ( 1 + --- (------)  )
 *                                                  2   |x+-1|
 *
 *                                              2
 *                                             y
 *                             =  |x+-1| + --------.
 *                                         2|x+-1|
 *
 *      Consequently, it is not difficult to see that
 *                                    2
 *                                   y
 *                     [ 1 + ------------ ,  if x < 1,
 *                     [      2(1+x)(1-x)
 *                     [
 *                     [
 *                     [ x,                 if x = 1 (y = 0),
 *                     [
 *              A ~=   [             2
 *                     [         x * y
 *                     [ x + ------------ ,  if x > 1
 *                     [      2(1+x)(x-1)
 *
 *      and hence
 *                        ______                                    2
```

```
 *                     / 2                    y              y
 *              A + \/ A  - 1  ~  1 + ---------------- + -----------, if x < 1,
 *                                   sqrt((x+1)(1-x))   2(x+1)(1-x)
 *
 *
 *                            ~  x + sqrt((x-1)*(x+1)),              if x >= 1.
 *
 *                                          2
 *                                         y
 *                           [ x(1 - ------------), if x < 1,
 *                           [       2(1+x)(1-x)
 *             B = x/A  ~  [
 *                           [ 1,                  if x = 1,
 *                           [
 *                           [           2
 *                           [          y
 *                           [ 1 - ------------ ,   if x > 1,
 *                           [      2(1+x)(1-x)
 *     Thus
 *                            [ asin(x) + i y/sqrt((x-1)*(x+1)), if x <  1
 *             casin(x+i*y)=[
 *                            [ pi/2    + i log(x+sqrt(x*x-1)),  if x >= 1
 *
 *  case 3. y < 4 sqrt(u), where u = minimum normal x.
 *      After case 1 and 2, this will only occurs when x=1. When x=1, we have
 *         A = (sqrt(4+y*y)+y)/2 ~ 1 + y/2 + y^2/8 + ...
 *      and
 *         B = 1/A = 1 - y/2 + y^2/8 + ...
 *      Since
 *         asin(x) = pi/2-2*asin(sqrt((1-x)/2))
 *         asin(x) = x + x^3/6 + x^5*3/40 + x^7*15/336 + ...
 *      we have, for the real part asin(B),
 *         asin(1-y/2) ~ pi/2 - 2 asin(sqrt(y/4))
 *                     ~ pi/2 - sqrt(y)
 *      For the imaginary part,
 *         log(A+sqrt(A*A-1)) ~ log(1+y/2+sqrt(2*y/2))
 *                            = log(1+y/2+sqrt(y))
 *                            = (y/2+sqrt(y)) - (y/2+sqrt(y))^2/2 + ...
 *                            ~ sqrt(y) - y*(sqrt(y)+y/2)/2
 *                            ~ sqrt(y)
 *
 *  case 4. y >= (x+1)ulp(0.5). In this case, A ~ y and B ~ x/y. Thus
 *         real part = asin(B) ~ x/y (be careful, x/y may underflow)
 *      and
 *         imag part = log(y+sqrt(y*y-one))
 *
 *
 *  case 5. Both x and y are large: x and y > sqrt(M)/8, where M = maximum x
 *      In this case,
 *         A ~ sqrt(x*x+y*y)
 *         B ~ x/sqrt(x*x+y*y).
 *      Thus
 *         real part = asin(B) = atan(x/y),
 *         imag part = log(A+sqrt(A*A-1)) ~ log(2A)
 *                   = log(2) + 0.5*log(x*x+y*y)
 *                   = log(2) + log(y) + 0.5*log(1+(x/y)^2)
 *
 *  case 6. x < 4 sqrt(u). In this case, we have
 *          A ~ sqrt(1+y*y), B = x/sqrt(1+y*y).
 *      Since B is tiny, we have
 *          real part = asin(B) ~ B = x/sqrt(1+y*y)
 *          imag part = log(A+sqrt(A*A-1)) = log (A+sqrt(y*y))
 *                    = log(y+sqrt(1+y*y))
 *                    = 0.5*log(y^2+2ysqrt(1+y^2)+1+y^2)
 *                    = 0.5*log(1+2y(y+sqrt(1+y^2)));
 *                    = 0.5*log1p(2y(y+A));
```



```
 *
 *      casin(z) = asin(B) + i sign(y) log (A + sqrt(A*A-1)),
 */
/* INDENT ON */

#include "libm.h"                       /* asin/atan/fabs/log/log1p/sqrt */
#include "complex_wrapper.h"

static const double zero = 0.0,
/* INDENT OFF */
static const double
        zero = 0.0,
        one = 1.0,
        E = 1.11022302462515654042e-16, /* 2**-53 */
        ln2 = 6.93147180559945286227e-01,
        pi_2 = 1.57079632679489656558e+00,
        pi_2_l = 6.12323399573676588 6e-17,
        pi_4 = 7.85398163397448278999e-01,
        Foursqrtu = 5.96667258496016539463e-154, /* 2**(-509) */
        Acrossover = 1.5,
        Bcrossover = 0.6417,
        half = 0.5;

/* INDENT ON */

dcomplex
casin(dcomplex z)
{
casin(dcomplex z) {
        double x, y, t, R, S, A, Am1, B, y2, xm1, xp1, Apx;
        int ix, iy, hx, hy;
        unsigned lx, ly;
        dcomplex ans;

        x = D_RE(z);
        y = D_IM(z);
        hx = HI_WORD(x);
        lx = LO_WORD(x);
        hy = HI_WORD(y);
        ly = LO_WORD(y);
        ix = hx & 0x7fffffff;
        iy = hy & 0x7fffffff;
        x = fabs(x);
        y = fabs(y);

        /* special cases */

        /* x is inf or NaN */
        if (ix >= 0x7ff00000) {         /* x is inf or NaN */
                if (ISINF(ix, lx)) {    /* x is INF */
                        D_IM(ans) = x;

#endif /* ! codereview */
                        if (iy >= 0x7ff00000) {
                                if (ISINF(iy, ly))
                                        /* casin(inf + i inf) = pi/4 + i inf */
                                        D_RE(ans) = pi_4;
                                else    /* casin(inf + i NaN) = NaN  + i inf  */
                                        D_RE(ans) = y + y;
                        } else { /* casin(inf + iy) = pi/2 + i inf */
                        } else  /* casin(inf + iy) = pi/2 + i inf */
                                D_RE(ans) = pi_2;
                        }
#endif /* ! codereview */
                } else {                /* x is NaN */
                        if (iy >= 0x7ff00000) {
```

```
233                                 /* INDENT OFF */
253                                 /*
254                                  * casin(NaN + i inf) = NaN + i inf
255                                  * casin(NaN + i NaN) = NaN + i NaN
256                                  */
238                                 /* INDENT ON */
257                                 D_IM(ans) = y + y;
258                                 D_RE(ans) = x + x;
259                         } else {
260                                 /* casin(NaN + i y ) = NaN  + i NaN */
261                                 D_IM(ans) = D_RE(ans) = x + y;
262                         }
263                 }

265 #endif /* ! codereview */
266                 if (hx < 0)
267                         D_RE(ans) = -D_RE(ans);

269 #endif /* ! codereview */
270                 if (hy < 0)
271                         D_IM(ans) = -D_IM(ans);

273 #endif /* ! codereview */
274                 return (ans);
275         }

277         /* casin(+0 + i 0  ) =  0   + i 0. */
278         if ((ix | lx | iy | ly) == 0)
279                 return (z);

281         if (iy >= 0x7ff00000) {         /* y is inf or NaN */
282                 if (ISINF(iy, ly)) {    /* casin(x + i inf) =  0   + i inf */
283                         D_IM(ans) = y;
284                         D_RE(ans) = zero;
285                 } else {                /* casin(x + i NaN) = NaN  + i NaN */
286                         D_IM(ans) = x + y;

288 #endif /* ! codereview */
289                         if ((ix | lx) == 0)
290                                 D_RE(ans) = x;
291                         else
292                                 D_RE(ans) = y;
293                 }

295 #endif /* ! codereview */
296                 if (hx < 0)
297                         D_RE(ans) = -D_RE(ans);

299 #endif /* ! codereview */
300                 if (hy < 0)
301                         D_IM(ans) = -D_IM(ans);

303 #endif /* ! codereview */
304                 return (ans);
305         }

307         if ((iy | ly) == 0) {           /* region 1: y=0 */
308                 if (ix < 0x3ff00000) {  /* |x| < 1 */
309                         D_RE(ans) = asin(x);
310                         D_IM(ans) = zero;
311                 } else {
312                         D_RE(ans) = pi_2;

314                         if (ix >= 0x43500000) {         /* |x| >= 2**54 */
246                         if (ix >= 0x43500000)   /* |x| >= 2**54 */
```



```
315                                 D_IM(ans) = ln2 + log(x);
316                         } else if (ix >= 0x3ff80000) {  /* x > Acrossover */
248                         else if (ix >= 0x3ff80000)      /* x > Acrossover */
317                                 D_IM(ans) = log(x + sqrt((x - one) * (x +
318                                     one)));
319                         } else {
251                         else {
320                                 xm1 = x - one;
321                                 D_IM(ans) = log1p(xm1 + sqrt(xm1 * (x + one)));
322                         }
323                 }
324         } else if (y <= E * fabs(x - one)) {    /* region 2: y < tiny*|x-1| */
325                 if (ix < 0x3ff00000) {          /* x < 1 */
326                         D_RE(ans) = asin(x);
327                         D_IM(ans) = y / sqrt((one + x) * (one - x));
328                 } else {
329                         D_RE(ans) = pi_2;

331                         if (ix >= 0x43500000)           /* |x| >= 2**54 */
262                         if (ix >= 0x43500000) { /* |x| >= 2**54 */
332                                 D_IM(ans) = ln2 + log(x);
333                         else if (ix >= 0x3ff80000)      /* x > Acrossover */
264                         } else if (ix >= 0x3ff80000)    /* x > Acrossover */
334                                 D_IM(ans) = log(x + sqrt((x - one) * (x +
335                                     one)));
336                         else
337                                 D_IM(ans) = log1p((x - one) + sqrt((x - one) *
338                                     (x + one)));
339                 }
340         } else if (y < Foursqrtu) {     /* region 3 */
341                 t = sqrt(y);
342                 D_RE(ans) = pi_2 - (t - pi_2_l);
343                 D_IM(ans) = t;
344         } else if (E * y - one >= x) { /* region 4 */
345                 D_RE(ans) = x / y; /* need to fix underflow cases */
346                 D_IM(ans) = ln2 + log(y);
347         } else if (ix >= 0x5fc00000 || iy >= 0x5fc00000) {      /* x,y>2**509 */
348                 /* region 5: x+1 or y is very large (>= sqrt(max)/8) */
349                 t = x / y;
350                 D_RE(ans) = atan(t);
351                 D_IM(ans) = ln2 + log(y) + half * log1p(t * t);
352         } else if (x < Foursqrtu) {
353                 /* region 6: x is very small, < 4sqrt(min) */
354                 A = sqrt(one + y * y);
355                 D_RE(ans) = x / A;      /* may underflow */

357 #endif /* ! codereview */
358                 if (iy >= 0x3ff80000)   /* if y > Acrossover */
359                         D_IM(ans) = log(y + A);
360                 else
361                         D_IM(ans) = half * log1p((y + y) * (y + A));
362         } else {                                /* safe region */
363                 y2 = y * y;
364                 xp1 = x + one;
365                 xm1 = x - one;
366                 R = sqrt(xp1 * xp1 + y2);
367                 S = sqrt(xm1 * xm1 + y2);
368                 A = half * (R + S);
369                 B = x / A;

371                 if (B <= Bcrossover) {
287                 if (B <= Bcrossover)
372                         D_RE(ans) = asin(B);
373                 } else {        /* use atan and an accurate approx to a-x */
289                 else {          /* use atan and an accurate approx to a-x */
374                         Apx = A + x;
```

```
#endif /* ! codereview */
                        if (x <= one)
                                D_RE(ans) = atan(x / sqrt(half * Apx * (y2 /
                                    (R + xp1) + (S - xm1))));
                        else
                                D_RE(ans) = atan(x / (y * sqrt(half * (Apx /
                                    (R + xp1) + Apx / (S + xm1)))));
                }

#endif /* ! codereview */
                if (A <= Acrossover) {
                        /* use log1p and an accurate approx to A-1 */
                        if (x < one)
                                Am1 = half * (y2 / (R + xp1) + y2 / (S - xm1));
                        else
                                Am1 = half * (y2 / (R + xp1) + (S + xm1));

#endif /* ! codereview */
                        D_IM(ans) = log1p(Am1 + sqrt(Am1 * (A + one)));
                } else {
                        D_IM(ans) = log(A + sqrt(A * A - one));
                }
        }

        if (hx < 0)
                D_RE(ans) = -D_RE(ans);

#endif /* ! codereview */
        if (hy < 0)
                D_IM(ans) = -D_IM(ans);

        return (ans);
}
```

```
**********************************************************
    1310 Thu Jun  6 20:39:34 2019
new/usr/src/lib/libm/common/complex/casinf.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __casinf = casinf

#include "libm.h"
#include "complex_wrapper.h"

fcomplex
casinf(fcomplex z)
{
casinf(fcomplex z) {
        dcomplex dz, dans;
        fcomplex ans;

        D_RE(dz) = (double)(F_RE(z));
        D_IM(dz) = (double)(F_IM(z));
        D_RE(dz) = (double) (F_RE(z));
        D_IM(dz) = (double) (F_IM(z));
        dans = casin(dz);
        F_RE(ans) = (float)(D_RE(dans));
        F_IM(ans) = (float)(D_IM(dans));
        F_RE(ans) = (float) (D_RE(dans));
        F_IM(ans) = (float) (D_IM(dans));
        return (ans);
}
______unchanged_portion_omitted_
```

```
**********************************************************
    1317 Thu Jun  6 20:39:35 2019
new/usr/src/lib/libm/common/complex/casinh.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __casinh = casinh


/* INDENT OFF */
/*
 * dcomplex casinh(dcomplex z);
 * casinh z = -i casin iz .
 */
/* INDENT ON */

#include "libm.h"
#include "complex_wrapper.h"

dcomplex
casinh(dcomplex z)
{
casinh(dcomplex z) {
        dcomplex w, r, ans;

        D_RE(w) = -D_IM(z);
        D_IM(w) = D_RE(z);
        r = casin(w);
        D_RE(ans) = D_IM(r);
        D_IM(ans) = -D_RE(r);
        return (ans);
}
______unchanged_portion_omitted_
```

```
**********************************************************
    1252 Thu Jun  6 20:39:35 2019
new/usr/src/lib/libm/common/complex/casinhf.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __casinhf = casinhf

#include "libm.h"
#include "complex_wrapper.h"

fcomplex
casinhf(fcomplex z)
{
casinhf(fcomplex z) {
        fcomplex w, r, ans;

        F_RE(w) = -F_IM(z);
        F_IM(w) = F_RE(z);
        r = casinf(w);
        F_RE(ans) = F_IM(r);
        F_IM(ans) = -F_RE(r);
        return (ans);
}
______unchanged_portion_omitted_
```

```
**********************************************************
    1263 Thu Jun  6 20:39:36 2019
new/usr/src/lib/libm/common/complex/casinhl.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __casinhl = casinhl

#include "libm.h"
#include "complex_wrapper.h"

ldcomplex
casinhl(ldcomplex z)
{
casinhl(ldcomplex z) {
        ldcomplex w, r, ans;

        LD_RE(w) = -LD_IM(z);
        LD_IM(w) = LD_RE(z);
        r = casinl(w);
        LD_RE(ans) = LD_IM(r);
        LD_IM(ans) = -LD_RE(r);
        return (ans);
}
______unchanged_portion_omitted_
```

```
**********************************************************
    6446 Thu Jun  6 20:39:36 2019
new/usr/src/lib/libm/common/complex/casinl.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
   1 /*
   2  * CDDL HEADER START
   3  *
   4  * The contents of this file are subject to the terms of the
   5  * Common Development and Distribution License (the "License").
   6  * You may not use this file except in compliance with the License.
   7  *
   8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
   9  * or http://www.opensolaris.org/os/licensing.
  10  * See the License for the specific language governing permissions
  11  * and limitations under the License.
  12  *
  13  * When distributing Covered Code, include this CDDL HEADER in each
  14  * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  15  * If applicable, add the following below this CDDL HEADER, with the
  16  * fields enclosed by brackets "[]" replaced with your own identifying
  17  * information: Portions Copyright [yyyy] [name of copyright owner]
  18  *
  19  * CDDL HEADER END
  20  */

  22 /*
  23  * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
  24  */

  26 #endif /* ! codereview */
  27 /*
  28  * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
  29  * Use is subject to license terms.
  30  */

  32 #pragma weak __casinl = casinl

  34 #include "libm.h"       /* asinl/atanl/fabsl/isinfl/log1pl/logl/sqrtl */
  35 #include "complex_wrapper.h"
  36 #include "longdouble.h"

  38 /* BEGIN CSTYLED */
  39 static const long double zero = 0.0L,
  40         one = 1.0L,
  41         Acrossover = 1.5L,
  42         Bcrossover = 0.6417L,
  43         half = 0.5L,
  44         ln2 = 6.931471805599453094172321214581765680755e-0001L,
  45         Foursqrtu = 7.334415470219388662485649568193932663825 5e-2466L,  /* 2**-8
  25 /* INDENT OFF */
  26 static const long double
  27 zero = 0.0L,
  28 one = 1.0L,
  29 Acrossover = 1.5L,
  30 Bcrossover = 0.6417L,
  31 half = 0.5L,
  32 ln2 = 6.931471805599453094172321214581765680755e-0001L,
  33 Foursqrtu = 7.334415470219388662485649568193932663825 5e-2466L,  /* 2**-8189 */
  46 #if defined(__x86)
  47         E = 5.421010862427522170037264004349708557 1289e-20L,    /* 2**-64 */
  48         pi_4 = 0.785398163397448309573992131227271329 4078130L,
  49         pi_4_l = 4.166871459260439164147932234267019303 6704898e-20L,
  50         pi_2 = 1.570796326794896619147984262454542658 8156260L,
  51         pi_2_l = 8.333742918520878328295864468534038607 3409796e-20L;
  35 E = 5.421010862427522170037264004349708557 1289e-20L,    /* 2**-64 */
```



```
  36 pi_4 = 0.785398163397448309573992131227271329 4078130L,
  37 pi_4_l = 4.166871459260439164147932234267019303 6704898e-20L,
  38 pi_2 = 1.570796326794896619147984262454542658 8156260L,
  39 pi_2_l = 8.333742918520878328295864468534038607 3409796e-20L;

  52 #else
  53         E = 9.629649721936179265279889712924636592 6905e-35L,    /* 2**-113 */
  54         pi_4 = 0.785398163397448309615660845819875699 3697670L,
  55         pi_4_l = 2.167952532530945256199261006510837992 1905808e-35L,
  56         pi_2 = 1.570796326794896619231321691639751398 7395340L,
  57         pi_2_l = 4.335905065061890512398522013021675984 3811616e-35L;
  42 E = 9.629649721936179265279889712924636592 6905e-35L,    /* 2**-113 */
  43 pi_4 = 0.785398163397448309615660845819875699 3697670L,
  44 pi_4_l = 2.167952532530945256199261006510837992 1905808e-35L,
  45 pi_2 = 1.570796326794896619231321691639751398 7395340L,
  46 pi_2_l = 4.335905065061890512398522013021675984 3811616e-35L;

  58 #endif
  59 /* END CSTYLED */
  49 /* INDENT ON */

  61 #if defined(__x86)
  62 static const int ip1 = 0x40400000;      /* 2**65 */
  63 #else
  64 static const int ip1 = 0x40710000;      /* 2**114 */
  65 #endif

  67 ldcomplex
  68 casinl(ldcomplex z)
  69 {
  58 casinl(ldcomplex z) {
  70         long double x, y, t, R, S, A, Am1, B, y2, xm1, xp1, Apx;
  71         int ix, iy, hx, hy;
  72         ldcomplex ans;

  74         x = LD_RE(z);
  75         y = LD_IM(z);
  76         hx = HI_XWORD(x);
  77         hy = HI_XWORD(y);
  78         ix = hx & 0x7fffffff;
  79         iy = hy & 0x7fffffff;
  80         x = fabsl(x);
  81         y = fabsl(y);

  83         /* special cases */

  85         /* x is inf or NaN */
  86         if (ix >= 0x7fff0000) {         /* x is inf or NaN */
  87                 if (isinfl(x)) {        /* x is INF */
  88                         LD_IM(ans) = x;

  90 #endif /* ! codereview */
  91                         if (iy >= 0x7fff0000) {
  92                                 if (isinfl(y))
  93                                         /* casin(inf + i inf) = pi/4 + i inf */
  94                                         LD_RE(ans) = pi_4 + pi_4_l;
  95                                 else    /* casin(inf + i NaN) = NaN + i inf */
  96                                         LD_RE(ans) = y + y;
  97                         } else { /* casin(inf + iy) = pi/2 + i inf */
  78                         } else  /* casin(inf + iy) = pi/2 + i inf */
  98                                 LD_RE(ans) = pi_2 + pi_2_l;
  99                         }
 100 #endif /* ! codereview */
 101                 } else {                /* x is NaN */
 102                         if (iy >= 0x7fff0000) {
```

```
                                /* INDENT OFF */
                                /*
                                 * casin(NaN + i inf) = NaN  + i inf
                                 * casin(NaN + i NaN) = NaN  + i NaN
                                 */
                                /* INDENT ON */
                                LD_IM(ans) = y + y;
                                LD_RE(ans) = x + x;
                        } else {
                                /*
                                 * casin(NaN + i y ) = NaN  + i NaN
                                 */
                                /* INDENT OFF */
                                /* casin(NaN + i y ) = NaN  + i NaN */
                                /* INDENT ON */
                                LD_IM(ans) = LD_RE(ans) = x + y;
                        }
                }

#endif /* ! codereview */
                if (hx < 0)
                        LD_RE(ans) = -LD_RE(ans);

#endif /* ! codereview */
                if (hy < 0)
                        LD_IM(ans) = -LD_IM(ans);

#endif /* ! codereview */
                return (ans);
        }

        /* casin(+0 + i 0) = 0 + i 0. */
        if (x == zero && y == zero)
                return (z);

        if (iy >= 0x7fff0000) {         /* y is inf or NaN */
                if (isinfl(y)) {        /* casin(x + i inf) = 0 + i inf */
                        LD_IM(ans) = y;
                        LD_RE(ans) = zero;
                } else {                /* casin(x + i NaN) = NaN + i NaN */
                        LD_IM(ans) = x + y;

#endif /* ! codereview */
                        if (x == zero)
                                LD_RE(ans) = x;
                        else
                                LD_RE(ans) = y;
                }

#endif /* ! codereview */
                if (hx < 0)
                        LD_RE(ans) = -LD_RE(ans);

#endif /* ! codereview */
                if (hy < 0)
                        LD_IM(ans) = -LD_IM(ans);

#endif /* ! codereview */
                return (ans);
        }

        if (y == zero) {                /* region 1: y=0 */
                if (ix < 0x3fff0000) {  /* |x| < 1 */
                        LD_RE(ans) = asinl(x);
                        LD_IM(ans) = zero;
                } else {
```



```
                        LD_RE(ans) = pi_2 + pi_2_l;

                        if (ix >= ip1) { /* |x| >= i386 ? 2**65 : 2**114 */
                        if (ix >= ip1)  /* |x| >= i386 ? 2**65 : 2**114 */
                                LD_IM(ans) = ln2 + logl(x);
                        } else if (ix >= 0x3fff8000) {  /* x > Acrossover */
                        else if (ix >= 0x3fff8000)      /* x > Acrossover */
                                LD_IM(ans) = logl(x + sqrtl((x - one) * (x +
                                    one)));
                        } else {
                        else {
                                xm1 = x - one;
                                LD_IM(ans) = log1pl(xm1 + sqrtl(xm1 * (x +
                                    one)));
                        }
                }
        } else if (y <= E * fabsl(x - one)) {   /* region 2: y < tiny*|x-1| */
                if (ix < 0x3fff0000) {          /* x < 1 */
                        LD_RE(ans) = asinl(x);
                        LD_IM(ans) = y / sqrtl((one + x) * (one - x));
                } else {
                        LD_RE(ans) = pi_2 + pi_2_l;

#endif /* ! codereview */
                        if (ix >= ip1) /* i386 ? 2**65 : 2**114 */
                                LD_IM(ans) = ln2 + logl(x);
                        else if (ix >= 0x3fff8000)      /* x > Acrossover */
                                LD_IM(ans) = logl(x + sqrtl((x - one) * (x +
                                    one)));
                        else
                                LD_IM(ans) = log1pl((x - one) + sqrtl((x -
                                    one) * (x + one)));
                }
        } else if (y < Foursqrtu) {     /* region 3 */
                t = sqrtl(y);
                LD_RE(ans) = pi_2 - (t - pi_2_l);
                LD_IM(ans) = t;
        } else if (E * y - one >= x) {  /* region 4 */
                LD_RE(ans) = x / y;     /* need to fix underflow cases */
                LD_IM(ans) = ln2 + logl(y);
        } else if (ix >= 0x5ffb0000 || iy >= 0x5ffb0000) {
                /* region 5: x+1 and y are both (>= sqrt(max)/8) i.e. 2**8188 */
                t = x / y;
                LD_RE(ans) = atanl(t);
                LD_IM(ans) = ln2 + logl(y) + half * log1pl(t * t);
        } else if (x < Foursqrtu) {
                /* region 6: x is very small, < 4sqrt(min) */
                A = sqrtl(one + y * y);
                LD_RE(ans) = x / A;     /* may underflow */

#endif /* ! codereview */
                if (iy >= 0x3fff8000)   /* if y > Acrossover */
                        LD_IM(ans) = logl(y + A);
                else
                        LD_IM(ans) = half * log1pl((y + y) * (y + A));
        } else {                        /* safe region */
                y2 = y * y;
                xp1 = x + one;
                xm1 = x - one;
                R = sqrtl(xp1 * xp1 + y2);
                S = sqrtl(xm1 * xm1 + y2);
                A = half * (R + S);
                B = x / A;

                if (B <= Bcrossover) {
                if (B <= Bcrossover)
```

```
                        LD_RE(ans) = asinl(B);
                } else {        /* use atan and an accurate approx to a-x */
                else {          /* use atan and an accurate approx to a-x */
                        Apx = A + x;

#endif /* ! codereview */
                        if (x <= one)
                                LD_RE(ans) = atanl(x / sqrtl(half * Apx * (y2 /
                                    (R + xp1) + (S - xm1))));
                        else
                                LD_RE(ans) = atanl(x / (y * sqrtl(half * (Apx /
                                    (R + xp1) + Apx / (S + xm1)))));
                }

#endif /* ! codereview */
                if (A <= Acrossover) {
                        /* use log1p and an accurate approx to A-1 */
                        if (x < one)
                                Am1 = half * (y2 / (R + xp1) + y2 / (S - xm1));
                        else
                                Am1 = half * (y2 / (R + xp1) + (S + xm1));

#endif /* ! codereview */
                        LD_IM(ans) = log1pl(Am1 + sqrtl(Am1 * (A + one)));
                } else {
                        LD_IM(ans) = logl(A + sqrtl(A * A - one));
                }
        }

        if (hx < 0)
                LD_RE(ans) = -LD_RE(ans);

#endif /* ! codereview */
        if (hy < 0)
                LD_IM(ans) = -LD_IM(ans);

        return (ans);
}
```

```
**********************************************************
    8774 Thu Jun  6 20:39:37 2019
new/usr/src/lib/libm/common/complex/catan.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __catan = catan


/* INDENT OFF */
/*
 * dcomplex catan(dcomplex z);
 *
 * If
 *     z = x + iy,
 *
 * then
 *          1       (    2x     )    1                 2    2
 * Re w  =  - arctan(-----------)  = - ATAN2(2x, 1 - x  - y )
 *          2       (     2    2)    2
 *                  (1 - x  - y )
 *
 *               ( 2         2)
 *          1    (x  +  (y+1) )      1                  4y
 * Im w  =  - log(------------) .=  --- log [ 1 + ------------- ]
 *          4    ( 2         2)      4                2         2
 *               (x  +  (y-1) )                     x  +  (y-1)
 *
 *                 2    16  3                         y
 *         = t - 2t   + -- t  - ..., where t = -----------------
 *                      3                       x*x + (y-1)*(y-1)
 *
 * Note that: if catan( x, y) = ( u, v), then
 *                catan(-x, y) = (-u, v)
 *                catan( x,-y) = ( u,-v)
 *
```



```
 * Also,   catan(x,y) = -i*catanh(-y,x), or
 *        catanh(x,y) =  i*catan(-y,x)
 * So, if catanh(y,x) = (v,u), then catan(x,y) = -i*(-v,u) = (u,v), i.e.,
 *        catan(x,y) = (u,v)
 *
 * EXCEPTION CASES (conform to ISO/IEC 9899:1999(E)):
 *    catan( 0  , 0   ) =  (0    ,  0   )
 *    catan( NaN, 0   ) =  (NaN  ,  0   )
 *    catan( 0  , 1   ) =  (0    ,  +inf) with divide-by-zero
 *    catan( inf, y   ) =  (pi/2 ,  0   ) for finite +y
 *    catan( NaN, y   ) =  (NaN  ,  NaN ) with invalid for finite y != 0
 *    catan( x  , inf ) =  (pi/2 ,  0   ) for finite +x
 *    catan( inf, inf ) =  (pi/2 ,  0   )
 *    catan( NaN, inf ) =  (NaN  ,  0   )
 *    catan( x  , NaN ) =  (NaN  ,  NaN ) with invalid for finite x
 *    catan( inf, NaN ) =  (pi/2 ,  +-0 )
 */
/* INDENT ON */

#include "libm.h"                       /* atan/atan2/fabs/log/log1p */
#include "complex_wrapper.h"

static const double pi_2 = 1.570796326794896558e+00,
/* INDENT OFF */
static const double
        pi_2 = 1.570796326794896558e+00,
        zero = 0.0,
        half = 0.5,
        two = 2.0,
        ln2 = 6.931471805599453094172321214581765680755e-0001,
        one = 1.0;

/* INDENT ON */

dcomplex
catan(dcomplex z)
{
catan(dcomplex z) {
        dcomplex ans;
        double x, y, ax, ay, t;
        int hx, hy, ix, iy;
        unsigned lx, ly;

        x = D_RE(z);
        y = D_IM(z);
        ax = fabs(x);
        ay = fabs(y);
        hx = HI_WORD(x);
        lx = LO_WORD(x);
        hy = HI_WORD(y);
        ly = LO_WORD(y);
        ix = hx & 0x7fffffff;
        iy = hy & 0x7fffffff;

        /* x is inf or NaN */
        if (ix >= 0x7ff00000) {
                if (ISINF(ix, lx)) {
                        D_RE(ans) = pi_2;
                        D_IM(ans) = zero;
                } else {
                        D_RE(ans) = x + x;

#endif /* ! codereview */
                        if ((iy | ly) == 0 || (ISINF(iy, ly)))
                                D_IM(ans) = zero;
                        else
```

```
121                                 D_IM(ans) = (fabs(y) - ay) / (fabs(y) - ay);
122                 }
123         } else if (iy >= 0x7ff00000) {
124                 /* y is inf or NaN */
125                 if (ISINF(iy, ly)) {
126                         D_RE(ans) = pi_2;
127                         D_IM(ans) = zero;
128                 } else {
129                         D_RE(ans) = (fabs(x) - ax) / (fabs(x) - ax);
130                         D_IM(ans) = y;
131                 }
132         } else if ((ix | lx) == 0) {
133                 /* BEGIN CSTYLED */
109                 /* INDENT OFF */
134                 /*
135                  * x = 0
136                  *      1                           1
137                  * A = --- * atan2(2x, 1-x*x-y*y) = --- atan2(0,1-|y|)
138                  *      2                           2
139                  *
140                  *     1     [  (y+1)*(y+1) ]   1          2      1         2y
141                  * B = - log [ ------------ ] = - log (1+ ---) or - log(1+ ----)
142                  *     4     [  (y-1)*(y-1) ]   2         y-1     2         1-y
143                  */
144                 /* END CSTYLED */
120                 /* INDENT ON */
145                 t = one - ay;

147 #endif /* ! codereview */
148                 if (((iy - 0x3ff00000) | ly) == 0) {
149                         /* y=1: catan(0,1)=(0,+inf) with 1/0 signal */
150                         D_IM(ans) = ay / ax;
151                         D_RE(ans) = zero;
152                 } else if (iy >= 0x3ff00000) {  /* y>1 */
153                         D_IM(ans) = half * log1p(two / (-t));
154                         D_RE(ans) = pi_2;
155                 } else {                        /* y<1 */
156                         D_IM(ans) = half * log1p((ay + ay) / t);
157                         D_RE(ans) = zero;
158                 }
159         } else if (iy < 0x3e200000 || ((ix - iy) >> 20) >= 30) {
160                 /* BEGIN CSTYLED */
122         /* INDENT OFF */
161                 /*
162                  * Tiny y (relative to 1+|x|)
163                  *     |y| < E*(1+|x|)
164                  * where E=2**-29, -35, -60 for double, double extended, quad pr
165                  *
166                  *      1                          [ x<=1:   atan(x)
167                  * A = --- * atan2(2x, 1-x*x-y*y) ~ [       1                  1+
168                  *      2                          [ x>=1: - atan2(2,(1-x)*(---
169                  *                                          2                  x
170                  *
171                  *                                   y/x
172                  * B ~ t*(1-2t), where t = ----------------- is tiny
173                  *                         x + (y-1)*(y-1)/x
174                  */
175                 /* END CSTYLED */
137                 /* INDENT ON */
176                 if (ix < 0x3ff00000)
177                         D_RE(ans) = atan(ax);
178                 else
179                         D_RE(ans) = half * atan2(two, (one - ax) * (one + one /
180                             ax));

141                         D_RE(ans) = half * atan2(two, (one - ax) * (one +
```



```
142                                 one / ax));
182                 if ((iy | ly) == 0) {
183                         D_IM(ans) = ay;
184                 } else {
185                         if (ix < 0x3e200000)
186                                 t = ay / ((ay - one) * (ay - one));
187                         else if (ix > 0x41c00000)
188                                 t = (ay / ax) / ax;
189                         else
190                                 t = ay / (ax * ax + (ay - one) * (ay - one));

192 #endif /* ! codereview */
193                         D_IM(ans) = t * (one - (t + t));
194                 }
195         } else if (iy >= 0x41c00000 && ((iy - ix) >> 20) >= 30) {
196                 /* BEGIN CSTYLED */
152                 /* INDENT OFF */
197                 /*
198                  * Huge y relative to 1+|x|
199                  *            |y| > Einv*(1+|x|), where Einv~2**(prec/2+3),
200                  *            1
201                  *       A ~ --- * atan2(2x, -y*y) ~ pi/2
202                  *            2
203                  *                                          y
204                  *       B ~ t*(1-2t), where t = --------------- is tiny
205                  *                                  (y-1)*(y-1)
206                  */
207                 /* END CSTYLED */
163                 /* INDENT ON */
208                 D_RE(ans) = pi_2;
209                 t = (ay / (ay - one)) / (ay - one);
210                 D_IM(ans) = t * (one - (t + t));
211         } else if (((iy - 0x3ff00000) | ly) == 0) {
212                 /* BEGIN CSTYLED */
168                 /* INDENT OFF */
213                 /*
214                  * y = 1
215                  *      1                           1
216                  * A = --- * atan2(2x, -x*x) = --- atan2(2,-x)
217                  *      2                           2
218                  *
219                  *     1     [x*x + 4]   1          4     [ 0.5(log2-logx) if
220                  * B = - log [-------] = - log (1+ ---) = [ |x|<E, else 0.25*
221                  *     4     [  x*x  ]   4         x*x    [ log1p((2/x)*(2/x))
222                  */
223                 /* END CSTYLED */
179                 /* INDENT ON */
224                 D_RE(ans) = half * atan2(two, -ax);

226                 if (ix < 0x3e200000) {
181                 if (ix < 0x3e200000)
227                         D_IM(ans) = half * (ln2 - log(ax));
228                 } else {
183                 else {
229                         t = two / ax;
230                         D_IM(ans) = 0.25 * log1p(t * t);
231                 }
232         } else if (ix >= 0x43900000) {
233                 /* BEGIN CSTYLED */
188                 /* INDENT OFF */
234                 /*
235                  * Huge x:
236                  * when |x| > 1/E^2,
237                  *      1                                pi
238                  * A ~ --- * atan2(2x, -x*x-y*y) ~ ---
239                  *      2                                2
```

```
 240                 *                                y                y/x
 241                 * B ~ t*(1-2t), where t = --------------- = (-------------- )/x
 242                 *                        x*x+(y-1)*(y-1)     1+((y-1)/x)^2
 243                 */
 244                /* END CSTYLED */
 199                /* INDENT ON */
 245                D_RE(ans) = pi_2;
 246                t = ((ay / ax) / (one + ((ay - one) / ax) * ((ay - one) /
 247                    ax))) / ax;
 248                D_IM(ans) = t * (one - (t + t));
 249        } else if (ix < 0x38b00000) {
 250                /* BEGIN CSTYLED */
 205                /* INDENT OFF */
 251                /*
 252                 * Tiny x:
 253                 * when |x| < E^4,  (note that y != 1)
 254                 *      1                          1
 255                 * A = --- * atan2(2x, 1-x*x-y*y) ~ --- * atan2(2x,(1-y)*(1+y))
 256                 *      2                          2
 257                 *
 258                 *     1     [(y+1)*(y+1)]   1          2      1          2y
 259                 * B = - log [-----------] = - log (1+ ---) or - log(1+ ----)
 260                 *     4     [(y-1)*(y-1)]   2         y-1     2         1-y
 261                 */
 262                /* END CSTYLED */
 217                /* INDENT ON */
 263                D_RE(ans) = half * atan2(ax + ax, (one - ay) * (one + ay));

 265 #endif /* ! codereview */
 266                if (iy >= 0x3ff00000)
 267                        D_IM(ans) = half * log1p(two / (ay - one));
 268                else
 269                        D_IM(ans) = half * log1p((ay + ay) / (one - ay));
 270        } else {
 271                /* BEGIN CSTYLED */
 219                /* INDENT OFF */
 272                /*
 273                 * normal x,y
 274                 *      1
 275                 * A = --- * atan2(2x, 1-x*x-y*y)
 276                 *      2
 277                 *
 278                 *     1     [x*x+(y+1)*(y+1)]   1               4y
 279                 * B = - log [---------------] = - log (1+ -----------------)
 280                 *     4     [x*x+(y-1)*(y-1)]   4         x*x + (y-1)*(y-1)
 281                 */
 282                /* END CSTYLED */
 230                /* INDENT ON */
 283                t = one - ay;

 285 #endif /* ! codereview */
 286                if (iy >= 0x3fe00000 && iy < 0x40000000) {
 287                        /* y close to 1 */
 288                        D_RE(ans) = half * (atan2((ax + ax), (t * (one + ay) -
 289                            ax * ax)));
 290                } else if (ix >= 0x3fe00000 && ix < 0x40000000) {
 291                        /* x close to 1 */
 292                        D_RE(ans) = half * atan2((ax + ax), ((one - ax) * (one +
 293                            ax) - ay * ay));
 294                } else {
 232                        D_RE(ans) = half * atan2((ax + ax), ((one - ax) *
 233                                (one + ax) - ay * ay));
 234                } else
 295                        D_RE(ans) = half * atan2((ax + ax), ((one - ax * ax) -
 296                            ay * ay));
 297                }
```



```
 299 #endif /* ! codereview */
 300                D_IM(ans) = 0.25 * log1p((4.0 * ay) / (ax * ax + t * t));
 301        }

 303 #endif /* ! codereview */
 304        if (hx < 0)
 305                D_RE(ans) = -D_RE(ans);

 307 #endif /* ! codereview */
 308        if (hy < 0)
 309                D_IM(ans) = -D_IM(ans);

 311 #endif /* ! codereview */
 312        return (ans);
 313 }
```

```
**********************************************************
    3592 Thu Jun  6 20:39:37 2019
new/usr/src/lib/libm/common/complex/catanf.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

#endif /* ! codereview */
/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2005 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __catanf = catanf

#include "libm.h"
#include "complex_wrapper.h"

#if defined(__i386) && !defined(__amd64)
extern int __swapRP(int);
#endif

static const float pi_2 = 1.570796326794896558e+00F,
static const float
        pi_2 = 1.570796326794896558e+00F,
        zero = 0.0F,
        half = 0.5F,
        two = 2.0F,
        one = 1.0F;

fcomplex
catanf(fcomplex z)
{
catanf(fcomplex z) {
        fcomplex ans;
        float x, y, ax, ay, t;
        double dx, dy, dt;
        int hx, hy, ix, iy;

        x = F_RE(z);
        y = F_IM(z);
        ax = fabsf(x);
```



```
        ay = fabsf(y);
        hx = THE_WORD(x);
        hy = THE_WORD(y);
        ix = hx & 0x7fffffff;
        iy = hy & 0x7fffffff;

        if (ix >= 0x7f800000) {         /* x is inf or NaN */
                if (ix == 0x7f800000) {
                        F_RE(ans) = pi_2;
                        F_IM(ans) = zero;
                } else {
                        F_RE(ans) = x * x;

#endif /* ! codereview */
                        if (iy == 0 || iy == 0x7f800000)
                                F_IM(ans) = zero;
                        else
                                F_IM(ans) = (fabsf(y) - ay) / (fabsf(y) - ay);
                }
        } else if (iy >= 0x7f800000) {  /* y is inf or NaN */
                if (iy == 0x7f800000) {
                        F_RE(ans) = pi_2;
                        F_IM(ans) = zero;
                } else {
                        F_RE(ans) = (fabsf(x) - ax) / (fabsf(x) - ax);
                        F_IM(ans) = y * y;
                }
        } else if (ix == 0) {
                /* BEGIN CSTYLED */
                /* INDENT OFF */
                /*
                 * x = 0
                 *      1                                1
                 * A = --- * atan2(2x, 1-x*x-y*y) = --- atan2(0,1-|y|)
                 *      2                                2
                 *
                 *     1     [ (y+1)*(y+1) ]   1          2      1         2y
                 * B = - log [ ----------- ] = - log (1+ ---) or - log(1+ ----)
                 *     4     [ (y-1)*(y-1) ]   2         y-1     2         1-y
                 */
                /* END CSTYLED */
                /* INDENT ON */
                t = one - ay;

#endif /* ! codereview */
                if (iy == 0x3f800000) {
                        /* y=1: catan(0,1)=(0,+inf) with 1/0 signal */
                        F_IM(ans) = ay / ax;
                        F_RE(ans) = zero;
                } else if (iy > 0x3f800000) {   /* y>1 */
                        F_IM(ans) = half * log1pf(two / (-t));
                        F_RE(ans) = pi_2;
                } else {                        /* y<1 */
                        F_IM(ans) = half * log1pf((ay + ay) / t);
                        F_RE(ans) = zero;
                }
        } else {
                /* BEGIN CSTYLED */
                /* INDENT OFF */
                /*
                 * use double precision x,y
                 *      1
                 * A = --- * atan2(2x, 1-x*x-y*y)
                 *      2
                 *
                 *     1     [ x*x+(y+1)*(y+1) ]   1               4y
```

```
122                  * B = - log [ --------------- ] = - log (1+ -----------------)
123                  *     4     [ x*x+(y-1)*(y-1) ]   4         x*x + (y-1)*(y-1)
124                  */
125                 /* END CSTYLED */

 74                 /* INDENT ON */
127 #if defined(__i386) && !defined(__amd64)
128                 int rp = __swapRP(fp_extended);
129 #endif
130                 dx = (double)ax;
131                 dy = (double)ay;
132                 F_RE(ans) = (float)(0.5 * atan2(dx + dx, 1.0 - dx * dx - dy *
133                     dy));
 80                 F_RE(ans) = (float)(0.5 * atan2(dx + dx,
 81                     1.0 - dx * dx - dy * dy));
134                 dt = dy - 1.0;
135                 F_IM(ans) = (float)(0.25 * log1p(4.0 * dy / (dx * dx + dt *
136                     dt)));

 83                 F_IM(ans) = (float)(0.25 * log1p(4.0 * dy /
 84                     (dx * dx + dt * dt)));
138 #if defined(__i386) && !defined(__amd64)
139                 if (rp != fp_extended)
140                         (void) __swapRP(rp);
141 #endif
142         }

144 #endif /* ! codereview */
145         if (hx < 0)
146                 F_RE(ans) = -F_RE(ans);

148 #endif /* ! codereview */
149         if (hy < 0)
150                 F_IM(ans) = -F_IM(ans);

152 #endif /* ! codereview */
153         return (ans);
154 }
```

```
**********************************************************
    1397 Thu Jun  6 20:39:38 2019
new/usr/src/lib/libm/common/complex/catanh.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __catanh = catanh


/* INDENT OFF */
/*
 * z := x + iy
 * catanh(z)    = -i catan(iz)
 *              = -i catan(-y+ix)
 *              = (Im(catan(-y+ix)), -Re(catan(-y+ix)))
 */
/* INDENT ON */

#include "libm.h"
#include "complex_wrapper.h"

dcomplex
catanh(dcomplex z)
{
catanh(dcomplex z) {
        double x, y;
        dcomplex ans, ct;

        x = D_RE(z);
        y = D_IM(z);
        D_RE(z) = -y;
        D_IM(z) = x;
        ct = catan(z);
        D_RE(ans) = D_IM(ct);
        D_IM(ans) = -D_RE(ct);
        return (ans);
```



```
}
______unchanged_portion_omitted_
```

```
**********************************************************
    1282 Thu Jun  6 20:39:38 2019
new/usr/src/lib/libm/common/complex/catanhf.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __catanhf = catanhf

#include "libm.h"
#include "complex_wrapper.h"

fcomplex
catanhf(fcomplex z)
{
catanhf(fcomplex z) {
        float x, y;
        fcomplex ans, ct;

        x = F_RE(z);
        y = F_IM(z);
        F_RE(z) = -y;
        F_IM(z) = x;
        ct = catanf(z);
        F_RE(ans) = F_IM(ct);
        F_IM(ans) = -F_RE(ct);
        return (ans);
}
______unchanged_portion_omitted_
```

```
**********************************************************
    1299 Thu Jun  6 20:39:39 2019
new/usr/src/lib/libm/common/complex/catanhl.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __catanhl = catanhl

#include "libm.h"
#include "complex_wrapper.h"

ldcomplex
catanhl(ldcomplex z)
{
catanhl(ldcomplex z) {
        long double x, y;
        ldcomplex ans, ct;

        x = LD_RE(z);
        y = LD_IM(z);
        LD_RE(z) = -y;
        LD_IM(z) = x;
        ct = catanl(z);
        LD_RE(ans) = LD_IM(ct);
        LD_IM(ans) = -LD_RE(ct);
        return (ans);
}
_____unchanged_portion_omitted_
```

```
**********************************************************
   10273 Thu Jun  6 20:39:39 2019
new/usr/src/lib/libm/common/complex/catanl.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __catanl = catanl


/* INDENT OFF */
/*
 * ldcomplex catanl(ldcomplex z);
 *
 * Atan(z) return A + Bi where,
 *                1
 *      A = --- * atan2(2x, 1-x*x-y*y)
 *            2
 *
 *            1      [ x*x + (y+1)*(y+1) ]   1               4y
 *       B = --- log [ ---------------- ] = - log (1+ -----------------)
 *            4      [ x*x + (y-1)*(y-1) ]   4         x*x + (y-1)*(y-1)
 *
 *                 2    16  3                          y
 *         = t - 2t   + -- t  - ..., where t = -----------------
 *                      3                       x*x + (y-1)*(y-1)
 * Proof:
 * Let w = atan(z=x+yi) = A + B i. Then tan(w) = z.
 * Since sin(w) = (exp(iw)-exp(-iw))/(2i), cos(w)=(exp(iw)+exp(-iw))/(2),
 * Let p = exp(iw), then z = tan(w) = ((p-1/p)/(p+1/p))/i, or
 * iz = (p*p-1)/(p*p+1), or, after simplification,
 *      p*p = (1+iz)/(1-iz)                                   ... (1)
 * LHS of (1) = exp(2iw) = exp(2i(A+Bi)) = exp(-2B)*exp(2iA)
 *            = exp(-2B)*(cos(2A)+i*sin(2A))                  ... (2)
 *              1-y+ix   (1-y+ix)*(1+y+ix)   1-x*x-y*y + 2xi
 * RHS of (1) = ------ = ---------------- = --------------- ... (3)
 *              1+y-ix    (1+y)**2 + x**2    (1+y)**2 + x**2
```



```
 *
 * Comparing the real and imaginary parts of (2) and (3), we have:
 *      cos(2A) : 1-x*x-y*y = sin(2A) : 2x
 * and hence
 *      tan(2A) = 2x/(1-x*x-y*y), or
 *      A = 0.5 * atan2(2x, 1-x*x-y*y)                       ... (4)
 *
 * For the imaginary part B, Note that |p*p| = exp(-2B), and
 *      |1+iz|   |i-z|   hypot(x,(y-1))
 *       |----| = |---| = --------------
 *      |1-iz|   |i+z|   hypot(x,(y+1))
 * Thus
 *                 x*x + (y+1)*(y+1)
 *      exp(4B) = -----------------, or
 *                 x*x + (y-1)*(y-1)
 *
 *            1     [x^2+(y+1)^2]   1             4y
 *       B =  - log [------------] = - log(1+ -------------)  ... (5)
 *            4     [x^2+(y-1)^2]   4         x^2+(y-1)^2
 *
 * QED.
 *
 * Note that: if catan( x, y) = ( u, v), then
 *               catan(-x, y) = (-u, v)
 *               catan( x,-y) = ( u,-v)
 *
 * Also,   catan(x,y) = -i*catanh(-y,x), or
 *        catanh(x,y) =  i*catan(-y,x)
 * So, if catanh(y,x) = (v,u), then catan(x,y) = -i*(-v,u) = (u,v), i.e.,
 *         catan(x,y) = (u,v)
 *
 * EXCEPTION CASES (conform to ISO/IEC 9899:1999(E)):
 *    catan( 0  , 0   ) =  (0    ,  0   )
 *    catan( NaN, 0   ) =  (NaN  ,  0   )
 *    catan( 0  , 1   ) =  (0    ,  +inf) with divide-by-zero
 *    catan( inf, y   ) =  (pi/2 ,  0   ) for finite +y
 *    catan( NaN, y   ) =  (NaN  ,  NaN ) with invalid for finite y != 0
 *    catan( x  , inf ) =  (pi/2 ,  0   ) for finite +x
 *    catan( inf, inf ) =  (pi/2 ,  0   )
 *    catan( NaN, inf ) =  (NaN  ,  0   )
 *    catan( x  , NaN ) =  (NaN  ,  NaN ) with invalid for finite x
 *    catan( inf, NaN ) =  (pi/2 ,  +-0 )
 */
/* INDENT ON */

#include "libm.h"       /* atan2l/atanl/fabsl/isinfl/iszerol/log1pl/logl */
#include "complex_wrapper.h"
#include "longdouble.h"

/* BEGIN CSTYLED */
static const long double zero = 0.0L,
        one = 1.0L,
        two = 2.0L,
        half = 0.5L,
        ln2 = 6.931471805599453094172321214581765680755e-0001L,
        pi_2 = 1.570796326794896619231321691639751442098584699687552910487472L,
/* INDENT OFF */
static const long double
zero = 0.0L,
one = 1.0L,
two = 2.0L,
half = 0.5L,
ln2 = 6.931471805599453094172321214581765680755e-0001L,
pi_2 = 1.570796326794896619231321691639751442098584699687552910487472L,
#if defined(__x86)
        E = 2.910383045673370361328125000000000000000e-11L,             /* 2**-3
```

```
        Einv = 3.435973836800000000000000000000000000e+10L;  /* 2**+35 */
E = 2.910383045673370361328125000000000000000e-11L,     /* 2**-35 */
Einv = 3.435973836800000000000000000000000000e+10L;  /* 2**+35 */
#else
        E = 8.673617379884035472059622406959533691406e-19L,             /* 2**-6
        Einv = 1.152921504606846976000000000000000000000e18L;           /* 2**+6
E = 8.673617379884035472059622406959533691406e-19L,     /* 2**-60 */
Einv = 1.152921504606846976000000000000000000000e18L;   /* 2**+60 */
#endif
/* END CSTYLED */
/* INDENT ON */

ldcomplex
catanl(ldcomplex z)
{
catanl(ldcomplex z) {
        ldcomplex ans;
        long double x, y, t1, ax, ay, t;
        int hx, hy, ix, iy;

        x = LD_RE(z);
        y = LD_IM(z);
        ax = fabsl(x);
        ay = fabsl(y);
        hx = HI_XWORD(x);
        hy = HI_XWORD(y);
        ix = hx & 0x7fffffff;
        iy = hy & 0x7fffffff;

        /* x is inf or NaN */
        if (ix >= 0x7fff0000) {
                if (isinfl(x)) {
                        LD_RE(ans) = pi_2;
                        LD_IM(ans) = zero;
                } else {
                        LD_RE(ans) = x + x;

#endif /* ! codereview */
                        if (iszerol(y) || (isinfl(y)))
                                LD_IM(ans) = zero;
                        else
                                LD_IM(ans) = (fabsl(y) - ay) / (fabsl(y) - ay);
                }
        } else if (iy >= 0x7fff0000) {
                /* y is inf or NaN */
                if (isinfl(y)) {
                        LD_RE(ans) = pi_2;
                        LD_IM(ans) = zero;
                } else {
                        LD_RE(ans) = (fabsl(x) - ax) / (fabsl(x) - ax);
                        LD_IM(ans) = y;
                }
        } else if (iszerol(x)) {
                /* BEGIN CSTYLED */
                /* INDENT OFF */
                /*
                 * x = 0
                 *      1                           1
                 * A = --- * atan2(2x, 1-x*x-y*y) = --- atan2(0,1-|y|)
                 *      2                           2
                 *
                 *     1     [ (y+1)*(y+1) ]   1          2      1         2y
                 * B = - log [ ----------- ] = - log (1+ ---) or - log(1+ ----)
                 *     4     [ (y-1)*(y-1) ]   2         y-1     2         1-y
                 */
                /* END CSTYLED */
```



```
                /* INDENT ON */
                t = one - ay;

#endif /* ! codereview */
                if (ay == one) {
                        /* y=1: catan(0,1)=(0,+inf) with 1/0 signal */
                        LD_IM(ans) = ay / ax;
                        LD_RE(ans) = zero;
                } else if (ay > one) {  /* y>1 */
                        LD_IM(ans) = half * log1pl(two / (-t));
                        LD_RE(ans) = pi_2;
                } else {                        /* y<1 */
                        LD_IM(ans) = half * log1pl((ay + ay) / t);
                        LD_RE(ans) = zero;
                }
        } else if (ay < E * (one + ax)) {
                /* BEGIN CSTYLED */
                /* INDENT OFF */
                /*
                 * Tiny y (relative to 1+|x|)
                 *     |y| < E*(1+|x|)
                 * where E=2**-29, -35, -60 for double, extended, quad precision
                 *
                 *     1                           [x<=1:   atan(x)
                 * A = - * atan2(2x,1-x*x-y*y) ~ [      1                1+x
                 *     2                           [x>=1: - atan2(2,(1-x)*(-----))
                 *                                        2                 x
                 *
                 *                                y/x
                 * B ~ t*(1-2t), where t = ----------------- is tiny
                 *                           x + (y-1)*(y-1)/x
                 *
                 *                                y
                 * (when x < 2**-60, t = ----------- )
                 *                          (y-1)*(y-1)
                 */
                /* END CSTYLED */
                if (ay == zero) {
                /* INDENT ON */
                if (ay == zero)
                        LD_IM(ans) = ay;
                } else {
                else {
                        t1 = ay - one;

#endif /* ! codereview */
                        if (ix < 0x3fc30000)
                                t = ay / (t1 * t1);
                        else if (ix > 0x403b0000)
                                t = (ay / ax) / ax;
                        else
                                t = ay / (ax * ax + t1 * t1);

#endif /* ! codereview */
                        LD_IM(ans) = t * (one - two * t);
                }

#endif /* ! codereview */
                if (ix < 0x3fff0000)
                        LD_RE(ans) = atanl(ax);
                else
                        LD_RE(ans) = half * atan2l(two, (one - ax) * (one +
                            one / ax));

        } else if (ay > Einv * (one + ax)) {
                /* BEGIN CSTYLED */
```

```
179                 /* INDENT OFF */
237                 /*
238                  * Huge y relative to 1+|x|
239                  *     |y| > Einv*(1+|x|), where Einv~2**(prec/2+3),
240                  *      1
241                  * A ~ --- * atan2(2x, -y*y) ~ pi/2
242                  *      2
243                  *                                y
244                  * B ~ t*(1-2t), where t = --------------- is tiny
245                  *                           (y-1)*(y-1)
246                  */
247                 /* END CSTYLED */
190                 /* INDENT ON */
248                 LD_RE(ans) = pi_2;
249                 t = (ay / (ay - one)) / (ay - one);
250                 LD_IM(ans) = t * (one - (t + t));
251         } else if (ay == one) {
252                 /* BEGIN CSTYLED */
195                 /* INDENT OFF */
253                 /*
254                  * y=1
255                  *     1                       1
256                  * A = - * atan2(2x, -x*x) = --- atan2(2,-x)
257                  *     2                       2
258                  *
259                  *     1     [ x*x+4]   1          4     [ 0.5(log2-logx) if
260                  * B = - log [ -----] = - log (1+ ---) = [ |x|<E, else 0.25*
261                  *     4     [  x*x ]   4         x*x    [ log1p((2/x)*(2/x))
262                  */
263                 /* END CSTYLED */
206                 /* INDENT ON */
264                 LD_RE(ans) = half * atan2l(two, -ax);

266                 if (ax < E) {
208                 if (ax < E)
267                         LD_IM(ans) = half * (ln2 - logl(ax));
268                 } else {
210                 else {
269                         t = two / ax;
270                         LD_IM(ans) = 0.25L * log1pl(t * t);
271                 }
272         } else if (ax > Einv * Einv) {
273                 /* BEGIN CSTYLED */
215                 /* INDENT OFF */
274                 /*
275                  * Huge x:
276                  * when |x| > 1/E^2,
277                  *      1                          pi
278                  * A ~ --- * atan2(2x, -x*x-y*y) ~ ---
279                  *      2                          2
280                  *                              y                y/x
281                  * B ~ t*(1-2t), where t = --------------- = (-------------- )/x
282                  *                         x*x+(y-1)*(y-1)     1+((y-1)/x)^2
283                  */
284                 /* END CSTYLED */
226                 /* INDENT ON */
285                 LD_RE(ans) = pi_2;
286                 t = ((ay / ax) / (one + ((ay - one) / ax) * ((ay - one) /
287                     ax))) / ax;
288                 LD_IM(ans) = t * (one - (t + t));
289         } else if (ax < E * E * E * E) {
290                 /* BEGIN CSTYLED */
232                 /* INDENT OFF */
291                 /*
292                  * Tiny x:
293                  * when |x| < E^4,  (note that y != 1)
```



```
294                  *      1                                1
295                  * A = --- * atan2(2x, 1-x*x-y*y) ~ --- * atan2(2x,1-y*y)
296                  *      2                                2
297                  *
298                  *     1     [ (y+1)*(y+1) ]   1          2      1          2y
299                  * B = - log [ ------------ ] = - log (1+ ---) or - log(1+ ----)
300                  *     4     [ (y-1)*(y-1) ]   2         y-1     2          1-y
301                  */
302                 /* END CSTYLED */
244                 /* INDENT ON */
303                 LD_RE(ans) = half * atan2l(ax + ax, (one - ay) * (one + ay));

305 #endif /* ! codereview */
306                 if (ay > one)           /* y>1 */
307                         LD_IM(ans) = half * log1pl(two / (ay - one));
308                 else                    /* y<1 */
309                         LD_IM(ans) = half * log1pl((ay + ay) / (one - ay));
310         } else {
311                 /* BEGIN CSTYLED */
246                 /* INDENT OFF */
312                 /*
313                  * normal x,y
314                  *      1
315                  * A = --- * atan2(2x, 1-x*x-y*y)
316                  *      2
317                  *
318                  *     1     [ x*x+(y+1)*(y+1) ]   1                4y
319                  * B = - log [ --------------- ] = - log (1+ -----------------)
320                  *     4     [ x*x+(y-1)*(y-1) ]   4          x*x + (y-1)*(y-1)
321                  */
322                 /* END CSTYLED */
257                 /* INDENT ON */
323                 t = one - ay;

325 #endif /* ! codereview */
326                 if (iy >= 0x3ffe0000 && iy < 0x40000000) {
327                         /* y close to 1 */
328                         LD_RE(ans) = half * (atan2l((ax + ax), (t * (one + ay) -
329                             ax * ax)));
259                         LD_RE(ans) = half * (atan2l((ax + ax), (t * (one +
260                                 ay) - ax * ax)));
330                 } else if (ix >= 0x3ffe0000 && ix < 0x40000000) {
331                         /* x close to 1 */
332                         LD_RE(ans) = half * atan2l((ax + ax), ((one - ax) *
333                             (one + ax) - ay * ay));
334                 } else {
335                         LD_RE(ans) = half * atan2l((ax + ax), ((one - ax * ax) -
336                             ay * ay));
337                 }

265                 } else
266                         LD_RE(ans) = half * atan2l((ax + ax), ((one - ax *
267                                 ax) - ay * ay));
339                 LD_IM(ans) = 0.25L * log1pl((4.0L * ay) / (ax * ax + t * t));
340         }

342 #endif /* ! codereview */
343         if (hx < 0)
344                 LD_RE(ans) = -LD_RE(ans);

346 #endif /* ! codereview */
347         if (hy < 0)
348                 LD_IM(ans) = -LD_IM(ans);

350 #endif /* ! codereview */
351         return (ans);
```

```
}
```

```
**********************************************************
    1389 Thu Jun  6 20:39:40 2019
new/usr/src/lib/libm/common/complex/ccos.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __ccos = ccos


/* INDENT OFF */
/*
 * dcomplex ccos(dcomplex z);
 *
 * z := x+iy; since ccos(iz) = cosh(z), we have
 * ccos(z)      = ccos((-1)*(-z)) = ccos(i*i*(-z))
 *              = ccosh(i*(-z)) = ccosh(i*(-x-yi))
 *              = ccosh(y-ix)
 */
/* INDENT ON */

#include "libm.h"
#include "complex_wrapper.h"

dcomplex
ccos(dcomplex z)
{
ccos(dcomplex z) {
        double x, y;

        x = D_RE(z);
        y = D_IM(z);
        D_RE(z) = y;
        D_IM(z) = -x;
        return (ccosh(z));
}
______unchanged_portion_omitted_
```

```
**********************************************************
    1199 Thu Jun  6 20:39:40 2019
new/usr/src/lib/libm/common/complex/ccosf.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __ccosf = ccosf

#include "libm.h"
#include "complex_wrapper.h"

fcomplex
ccosf(fcomplex z)
{
ccosf(fcomplex z) {
        float x, y;

        x = F_RE(z);
        y = F_IM(z);
        F_RE(z) = y;
        F_IM(z) = -x;
        return (ccoshf(z));
}
______unchanged_portion_omitted_
```

```
**********************************************************
    3864 Thu Jun  6 20:39:41 2019
new/usr/src/lib/libm/common/complex/ccosh.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __ccosh = ccosh


/* INDENT OFF */
/*
 * dcomplex ccosh(dcomplex z);
 *
 *             z      -z       x                      -x
 *            e   +  e        e  (cos(y)+i*sin(y)) + e  (cos(-y)+i*sin(-y))
 * cosh z = -------------- =  ---------------------------------------------
 *                2                              2
 *                          x    -x                x    -x
 *            cos(y) ( e  + e  )  + i*sin(y) (e  - e   )
 *        = ---------------------------------------------
 *                                2
 *
 *        =  cos(y) cosh(x)  + i sin(y) sinh(x)
 *
 * Implementation Note
 * -------------------
 *
 *             |x|    -|x|   |x|        -2|x|       -2|x|    -P-4
 * Note that  e   +- e    = e   ( 1 +- e     ). If e      < 2     , where
 *
 * P stands for the number of significant bits of the machine precision,
 *                                      |x|
 * then the result will be rounded to e   . Therefore, we have
 *
 *                 z
 *                e
```



```
 *     cosh z = -----  if |x| >= (P/2 + 2)*ln2
 *                2
 *
 * EXCEPTION (conform to ISO/IEC 9899:1999(E)):
 *      ccosh(0,0)=(1,0)
 *      ccosh(0,inf)=(NaN,+-0)
 *      ccosh(0,NaN)=(NaN,+-0)
 *      ccosh(x,inf) = (NaN,NaN) for finite non-zero x
 *      ccosh(x,NaN) = (NaN,NaN) for finite non-zero x
 *      ccosh(inf,0) = (inf, 0)
 *      ccosh(inf,y) = (inf*cos(y),inf*sin(y)) for finite non-zero y
 *      ccosh(inf,inf) = (+-inf,NaN)
 *      ccosh(inf,NaN) = (+inf,NaN)
 *      ccosh(NaN,0) = (NaN,+-0)
 *      ccosh(NaN,y) = (NaN,NaN) for non-zero y
 *      ccosh(NaN,NaN) = (NaN,NaN)
 */
/* INDENT ON */

#include "libm.h"       /* cosh/exp/fabs/scalbn/sinh/sincos/__k_cexp */
#include "complex_wrapper.h"

dcomplex
ccosh(dcomplex z)
{
ccosh(dcomplex z) {
        double t, x, y, S, C;
        int hx, ix, lx, hy, iy, ly, n;
        dcomplex ans;

        x = D_RE(z);
        y = D_IM(z);
        hx = HI_WORD(x);
        lx = LO_WORD(x);
        ix = hx & 0x7fffffff;
        hy = HI_WORD(y);
        ly = LO_WORD(y);
        iy = hy & 0x7fffffff;
        x = fabs(x);
        y = fabs(y);

        (void) sincos(y, &S, &C);

#endif /* ! codereview */
        if (ix >= 0x403c0000) {         /* |x| > 28 = prec/2 (14,28,34,60) */
                if (ix >= 0x40862E42) { /* |x| > 709.78... ~ log(2**1024) */
                        if (ix >= 0x7ff00000) { /* |x| is inf or NaN */
                                if ((iy | ly) == 0) {
                                        D_RE(ans) = x;
                                        D_IM(ans) = y;
                                } else if (iy >= 0x7ff00000) {
                                        D_RE(ans) = x;
                                        D_IM(ans) = x - y;
                                } else {
                                        D_RE(ans) = C * x;
                                        D_IM(ans) = S * x;
                                }
                        } else {
                                t = __k_cexp(x, &n);
                                /* return exp(x)=t*2**n */
                                D_RE(ans) = scalbn(C * t, n - 1);
                                D_IM(ans) = scalbn(S * t, n - 1);
                        }
                } else {
                        t = exp(x) * 0.5;
                        D_RE(ans) = C * t;
```

```
                        D_IM(ans) = S * t;
                }
        } else {
                if ((ix | lx) == 0) {   /* x = 0, return (C,0) */
                        D_RE(ans) = C;
                        D_IM(ans) = 0.0;
                } else {
                        D_RE(ans) = C * cosh(x);
                        D_IM(ans) = S * sinh(x);
                }
        }

#endif /* ! codereview */
        if ((hx ^ hy) < 0)
                D_IM(ans) = -D_IM(ans);

#endif /* ! codereview */
        return (ans);
}
```

```
**********************************************************
    2527 Thu Jun  6 20:39:41 2019
new/usr/src/lib/libm/common/complex/ccoshf.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

#endif /* ! codereview */
/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2005 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __ccoshf = ccoshf

#include "libm.h"
#include "complex_wrapper.h"

#if defined(__i386) && !defined(__amd64)
extern int __swapRP(int);
#endif

static const float zero = 0.0F, half = 0.5F;

fcomplex
ccoshf(fcomplex z)
{
ccoshf(fcomplex z) {
        float t, x, y, S, C;
        double w;
        int hx, ix, hy, iy, n;
        fcomplex ans;

        x = F_RE(z);
        y = F_IM(z);
        hx = THE_WORD(x);
        ix = hx & 0x7fffffff;
        hy = THE_WORD(y);
        iy = hy & 0x7fffffff;
        x = fabsf(x);
        y = fabsf(y);
```



```
        sincosf(y, &S, &C);

#endif /* ! codereview */
        if (ix >= 0x41600000) {         /* |x| > 14 = prec/2 (14,28,34,60) */
                if (ix >= 0x42B171AA) { /* |x| > 88.722... ~ log(2**128) */
                        if (ix >= 0x7f800000) { /* |x| is inf or NaN */
                                if (iy == 0) {
                                        F_RE(ans) = x;
                                        F_IM(ans) = y;
                                } else if (iy >= 0x7f800000) {
                                        F_RE(ans) = x;
                                        F_IM(ans) = x - y;
                                } else {
                                        F_RE(ans) = C * x;
                                        F_IM(ans) = S * x;
                                }
                        } else {
#if defined(__i386) && !defined(__amd64)
                                int rp = __swapRP(fp_extended);
#endif
                                /* return (C, S) * exp(x) / 2 */
                                w = __k_cexp((double)x, &n);
                                F_RE(ans) = (float)scalbn(C * w, n - 1);
                                F_IM(ans) = (float)scalbn(S * w, n - 1);
#if defined(__i386) && !defined(__amd64)
                                if (rp != fp_extended)
                                        (void) __swapRP(rp);
#endif
                        }
                } else {
                        t = expf(x) * half;
                        F_RE(ans) = C * t;
                        F_IM(ans) = S * t;
                }
        } else {
                if (ix == 0) {          /* x = 0, return (C,0) */
                        F_RE(ans) = C;
                        F_IM(ans) = zero;
                } else {
                        F_RE(ans) = C * coshf(x);
                        F_IM(ans) = S * sinhf(x);
                }
        }

#endif /* ! codereview */
        if ((hx ^ hy) < 0)
                F_IM(ans) = -F_IM(ans);

#endif /* ! codereview */
        return (ans);
}
```

```
************************************************************
    2334 Thu Jun  6 20:39:42 2019
new/usr/src/lib/libm/common/complex/ccoshl.c
11210 libm should be cstyle(10NBLD) clean
************************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __ccoshl = ccoshl

#include "libm.h"       /* coshl/expl/fabsl/scalbnl/sincosl/sinhl/__k_cexpl */
#include "complex_wrapper.h"

/* INDENT OFF */
static const long double zero = 0.0L, half = 0.5L;

/* INDENT ON */

ldcomplex
ccoshl(ldcomplex z)
{
ccoshl(ldcomplex z) {
        long double t, x, y, S, C;
        int hx, ix, hy, iy, n;
        ldcomplex ans;

        x = LD_RE(z);
        y = LD_IM(z);
        hx = HI_XWORD(x);
        ix = hx & 0x7fffffff;
        hy = HI_XWORD(y);
        iy = hy & 0x7fffffff;
        x = fabsl(x);
        y = fabsl(y);

        (void) sincosl(y, &S, &C);

#endif /* ! codereview */
```



```
        if (ix >= 0x4004e000) {         /* |x| > 60 = prec/2 (14,28,34,60) */
                if (ix >= 0x400C62E4) { /* |x| > 11356.52... ~ log(2**16384) */
                        if (ix >= 0x7fff0000) { /* |x| is inf or NaN */
                                if (y == zero) {
                                        LD_RE(ans) = x;
                                        LD_IM(ans) = y;
                                } else if (iy >= 0x7fff0000) {
                                        LD_RE(ans) = x;
                                        LD_IM(ans) = x - y;
                                } else {
                                        LD_RE(ans) = C * x;
                                        LD_IM(ans) = S * x;
                                }
                        } else {
                                t = __k_cexpl(x, &n);
                                /* return exp(x)=t*2**n */
                                LD_RE(ans) = scalbnl(C * t, n - 1);
                                LD_IM(ans) = scalbnl(S * t, n - 1);
                        }
                } else {
                        t = expl(x) * half;
                        LD_RE(ans) = C * t;
                        LD_IM(ans) = S * t;
                }
        } else {
                if (x == zero) {        /* x = 0, return (C,0) */
                        LD_RE(ans) = C;
                        LD_IM(ans) = zero;
                } else {
                        LD_RE(ans) = C * coshl(x);
                        LD_IM(ans) = S * sinhl(x);
                }
        }

#endif /* ! codereview */
        if ((hx ^ hy) < 0)
                LD_IM(ans) = -LD_IM(ans);

#endif /* ! codereview */
        return (ans);
}
```

```
**********************************************************
    1211 Thu Jun  6 20:39:42 2019
new/usr/src/lib/libm/common/complex/ccosl.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __ccosl = ccosl

#include "libm.h"
#include "complex_wrapper.h"

ldcomplex
ccosl(ldcomplex z)
{
ccosl(ldcomplex z) {
        long double x, y;

        x = LD_RE(z);
        y = LD_IM(z);
        LD_RE(z) = y;
        LD_IM(z) = -x;
        return (ccoshl(z));
}
______unchanged_portion_omitted_
```

```
**********************************************************
    3054 Thu Jun  6 20:39:43 2019
new/usr/src/lib/libm/common/complex/cexp.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __cexp = cexp


/* INDENT OFF */
/*
 * dcomplex cexp(dcomplex z);
 *
 *  x+iy    x
 * e     = e  (cos(y)+i*sin(y))
 *
 * Over/underflow issue
 * --------------------
 * exp(x) may be huge but cos(y) or sin(y) may be tiny. So we use
 * function __k_cexp(x,&n) to return exp(x) = __k_cexp(x,&n)*2**n.
 * Thus if exp(x+iy) = A + Bi and t = __k_cexp(x,&n), then
 *         A = t*cos(y)*2**n,   B = t*sin(y)*2**n
 *
 * Purge off all exceptional arguments:
 *      (x,0) --> (exp(x),0)         for all x, include inf and NaN
 *      (+inf, y) --> (+inf, NaN)    for inf, nan
 *      (-inf, y) --> (+-0, +-0)     for y = inf, nan
 *      (x,+-inf/NaN) --> (NaN,NaN)  for finite x
 * For all other cases, return
 *      (x,y) --> exp(x)*cos(y)+i*exp(x)*sin(y))
 *
 * Algorithm for out of range x and finite y
 *      1. compute exp(x) in factor form (t=__k_cexp(x,&n))*2**n
 *      2. compute sincos(y,&s,&c)
 *      3. compute t*s+i*(t*c), then scale back to 2**n and return.
 */
```



```
/* INDENT ON */

#include "libm.h"                       /* exp/scalbn/sincos/__k_cexp */
#include "complex_wrapper.h"

static const double zero = 0.0;

dcomplex
cexp(dcomplex z)
{
cexp(dcomplex z) {
        dcomplex ans;
        double x, y, t, c, s;
        int n, ix, iy, hx, hy, lx, ly;

        x = D_RE(z);
        y = D_IM(z);
        hx = HI_WORD(x);
        lx = LO_WORD(x);
        hy = HI_WORD(y);
        ly = LO_WORD(y);
        ix = hx & 0x7fffffff;
        iy = hy & 0x7fffffff;

#endif /* ! codereview */
        if ((iy | ly) == 0) {           /* y = 0 */
                D_RE(ans) = exp(x);
                D_IM(ans) = y;
        } else if (ISINF(ix, lx)) {     /* x is +-inf */
                if (hx < 0) {
                        if (iy >= 0x7ff00000) {
                                D_RE(ans) = zero;
                                D_IM(ans) = zero;
                        } else {
                                sincos(y, &s, &c);
                                D_RE(ans) = zero * c;
                                D_IM(ans) = zero * s;
                        }
                } else {
                        if (iy >= 0x7ff00000) {
                                D_RE(ans) = x;
                                D_IM(ans) = y - y;
                        } else {
                                (void) sincos(y, &s, &c);
                                D_RE(ans) = x * c;
                                D_IM(ans) = x * s;
                        }
                }
        } else {
                (void) sincos(y, &s, &c);

#endif /* ! codereview */
                if (ix >= 0x40862E42) { /* |x| > 709.78... ~ log(2**1024) */
                        t = __k_cexp(x, &n);
                        D_RE(ans) = scalbn(t * c, n);
                        D_IM(ans) = scalbn(t * s, n);
                } else {
                        t = exp(x);
                        D_RE(ans) = t * c;
                        D_IM(ans) = t * s;
                }
        }

#endif /* ! codereview */
        return (ans);
}
```

```
**********************************************************
    2359 Thu Jun  6 20:39:43 2019
new/usr/src/lib/libm/common/complex/cexpf.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

#endif /* ! codereview */
/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2005 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __cexpf = cexpf

#include "libm.h"
#include "complex_wrapper.h"

#if defined(__i386) && !defined(__amd64)
extern int __swapRP(int);
#endif

static const float zero = 0.0F;

fcomplex
cexpf(fcomplex z)
{
cexpf(fcomplex z) {
        fcomplex ans;
        float x, y, c, s;
        double t;
        int n, ix, iy, hx, hy;

        x = F_RE(z);
        y = F_IM(z);
        hx = THE_WORD(x);
        hy = THE_WORD(y);
        ix = hx & 0x7fffffff;
        iy = hy & 0x7fffffff;

#endif /* ! codereview */
        if (iy == 0) {                  /* y = 0 */
```



```
                F_RE(ans) = expf(x);
                F_IM(ans) = y;
        } else if (ix == 0x7f800000) {  /* x is +-inf */
                if (hx < 0) {
                        if (iy >= 0x7f800000) {
                                F_RE(ans) = zero;
                                F_IM(ans) = zero;
                        } else {
                                sincosf(y, &s, &c);
                                F_RE(ans) = zero * c;
                                F_IM(ans) = zero * s;
                        }
                } else {
                        if (iy >= 0x7f800000) {
                                F_RE(ans) = x;
                                F_IM(ans) = y - y;
                        } else {
                                sincosf(y, &s, &c);
                                F_RE(ans) = x * c;
                                F_IM(ans) = x * s;
                        }
                }
        } else {
                sincosf(y, &s, &c);

#endif /* ! codereview */
                if (ix >= 0x42B171AA) { /* |x| > 88.722... ~ log(2**128) */
#if defined(__i386) && !defined(__amd64)
                        int rp = __swapRP(fp_extended);
#endif
                        t = __k_cexp(x, &n);
                        F_RE(ans) = (float)scalbn(t * (double)c, n);
                        F_IM(ans) = (float)scalbn(t * (double)s, n);
#if defined(__i386) && !defined(__amd64)
                        if (rp != fp_extended)
                                (void) __swapRP(rp);
#endif
                } else {
                        t = expf(x);
                        F_RE(ans) = t * c;
                        F_IM(ans) = t * s;
                }
        }

#endif /* ! codereview */
        return (ans);
}
```

```
**********************************************************
    2221 Thu Jun  6 20:39:43 2019
new/usr/src/lib/libm/common/complex/cexpl.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __cexpl = cexpl

#include "libm.h"       /* expl/isinfl/iszerol/scalbnl/sincosl */
#include "complex_wrapper.h"

extern int isinfl(long double);
extern int iszerol(long double);

/* INDENT OFF */
static const long double zero = 0.0L;

/* INDENT ON */

ldcomplex
cexpl(ldcomplex z)
{
cexpl(ldcomplex z) {
        ldcomplex ans;
        long double x, y, t, c, s;
        int n, ix, iy, hx, hy;

        x = LD_RE(z);
        y = LD_IM(z);
        hx = HI_XWORD(x);
        hy = HI_XWORD(y);
        ix = hx & 0x7fffffff;
        iy = hy & 0x7fffffff;

#endif /* ! codereview */
        if (iszerol(y)) {               /* y = 0 */
```



```
                LD_RE(ans) = expl(x);
                LD_IM(ans) = y;
        } else if (isinfl(x)) {         /* x is +-inf */
                if (hx < 0) {
                        if (iy >= 0x7fff0000) {
                                LD_RE(ans) = zero;
                                LD_IM(ans) = zero;
                        } else {
                                sincosl(y, &s, &c);
                                LD_RE(ans) = zero * c;
                                LD_IM(ans) = zero * s;
                        }
                } else {
                        if (iy >= 0x7fff0000) {
                                LD_RE(ans) = x;
                                LD_IM(ans) = y - y;
                        } else {
                                (void) sincosl(y, &s, &c);
                                LD_RE(ans) = x * c;
                                LD_IM(ans) = x * s;
                        }
                }
        } else {
                (void) sincosl(y, &s, &c);

#endif /* ! codereview */
                if (ix >= 0x400C62E4) { /* |x| > 11356.52... ~ log(2**16384) */
                        t = __k_cexpl(x, &n);
                        LD_RE(ans) = scalbnl(t * c, n);
                        LD_IM(ans) = scalbnl(t * s, n);
                } else {
                        t = expl(x);
                        LD_RE(ans) = t * c;
                        LD_IM(ans) = t * s;
                }
        }

#endif /* ! codereview */
        return (ans);
}
```

```
**********************************************************
    1124 Thu Jun  6 20:39:44 2019
new/usr/src/lib/libm/common/complex/cimag.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __cimag = cimag

#include "libm.h"
#include "complex_wrapper.h"

double
cimag(dcomplex z)
{
cimag(dcomplex z) {
        return (D_IM(z));
}
______unchanged_portion_omitted_
```

```
**********************************************************
    1126 Thu Jun  6 20:39:44 2019
new/usr/src/lib/libm/common/complex/cimagf.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __cimagf = cimagf

#include "libm.h"
#include "complex_wrapper.h"

float
cimagf(fcomplex z)
{
cimagf(fcomplex z) {
        return (F_IM(z));
}
______unchanged_portion_omitted_
```

```
**********************************************************
    1134 Thu Jun  6 20:39:45 2019
new/usr/src/lib/libm/common/complex/cimagl.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __cimagl = cimagl

#include "libm.h"
#include "complex_wrapper.h"

long double
cimagl(ldcomplex z)
{
cimagl(ldcomplex z) {
        return (LD_IM(z));
}
______unchanged_portion_omitted_
```

```
**********************************************************
    3751 Thu Jun  6 20:39:45 2019
new/usr/src/lib/libm/common/complex/clog.c
11210 libm should be cstyle(10NBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

#endif /* ! codereview */
/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2005 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak clog = __clog


/* INDENT OFF */
/*
 * dcomplex clog(dcomplex z);
 *
 *                          _______
 *                        / 2    2            -1   y
 * log(x+iy) = log(\/ x  + y    ) + i tan   (---)
 *                                            x
 *
 *              1       2    2         -1   y
 *           = --- log(x  + y ) + i tan   (---)
 *              2                         x
 *
 * Note that the arctangent ranges from -PI to +PI, thus the imaginary
 * part of clog is atan2(y,x).
 *
 * EXCEPTION CASES (conform to ISO/IEC 9899:1999(E)):
 *    clog(-0 + i 0   ) =  -inf + i pi
 *    clog( 0 + i 0   ) =  -inf + i 0
 *    clog( x + i inf ) =  -inf + i pi/2, for finite x
 *    clog( x + i NaN ) =  NaN  + i NaN with invalid for finite x
 *    clog(-inf + iy   )=  +inf + i pi, for finite positive-signed y
 *    clog(+inf + iy   )=  +inf + i 0 , for finite positive-signed y
 *    clog(-inf + i inf)=  inf  + i 3pi/4
 *    clog(+inf + i inf)=  inf  + i pi/4
 *    clog(+-inf+ i NaN)=  inf  + i NaN
```



```
 *    clog(NaN  + i y  )=  NaN  + i NaN for finite y
 *    clog(NaN  + i inf)=  inf  + i NaN
 *    clog(NaN  + i NaN)=  NaN  + i NaN
 */
/* INDENT ON */

#include <math.h>                       /* atan2/fabs/log/log1p */
#include "complex_wrapper.h"
#include "libm_protos.h"                /* __k_clog_r */


static const double half = 0.5, one = 1.0;

dcomplex
__clog(dcomplex z)
{
__clog(dcomplex z) {
        dcomplex ans;
        double x, y, t, ax, ay, w;
        int n, ix, iy, hx, hy;
        unsigned lx, ly;

        x = D_RE(z);
        y = D_IM(z);
        hx = HI_WORD(x);
        lx = LO_WORD(x);
        hy = HI_WORD(y);
        ly = LO_WORD(y);
        ix = hx & 0x7fffffff;
        iy = hy & 0x7fffffff;
        ay = fabs(y);
        ax = fabs(x);
        D_IM(ans) = carg(z);

#endif /* ! codereview */
        if (ix < iy || (ix == iy && lx < ly)) {
                /* swap x and y to force ax >= ay */
                t = ax;
                ax = ay;
                ay = t;
                n = ix, ix = iy;
                iy = n;
                n = lx, lx = ly;
                ly = n;
        }

#endif /* ! codereview */
        n = (ix - iy) >> 20;

#endif /* ! codereview */
        if (ix >= 0x7ff00000) {         /* x or y is Inf or NaN */
                if (ISINF(ix, lx))
                        D_RE(ans) = ax;
                else if (ISINF(iy, ly))
                        D_RE(ans) = ay;
                else
                        D_RE(ans) = ax * ay;
        } else if ((iy | ly) == 0) {
                D_RE(ans) = ((ix | lx) == 0) ? -one / ax : log(ax);
                D_RE(ans) = ((ix | lx) == 0)? -one / ax : log(ax);
        } else if (((0x3fffffff - ix) ^ (ix - 0x3fe00000)) >= 0) {
                /* 0.5 <= x < 2 */
                if (ix >= 0x3ff00000) {
                        if (((ix - 0x3ff00000) | lx) == 0)
                                D_RE(ans) = half * log1p(ay * ay);
                        else if (n >= 60)
```

```
                                D_RE(ans) = log(ax);
                        else
                                D_RE(ans) = half * (log1p(ay * ay + (ax - one) *
                                    (ax + one)));
                                D_RE(ans) = half * (log1p(ay * ay + (ax -
                                    one) * (ax + one)));
                } else if (n >= 60) {
                        D_RE(ans) = log(ax);
                } else {
                        D_RE(ans) = __k_clog_r(ax, ay, &w);
                }
        } else if (n >= 30) {
                D_RE(ans) = log(ax);
        } else if (ix < 0x5f300000 && iy >= 0x20b00000) {
                /* 2**-500< y < x < 2**500 */
                D_RE(ans) = half * log(ax * ax + ay * ay);
        } else {
                t = ay / ax;
                D_RE(ans) = log(ax) + half * log1p(t * t);
        }

#endif /* ! codereview */
        return (ans);
}
```

```
**********************************************************
    2132 Thu Jun  6 20:39:46 2019
new/usr/src/lib/libm/common/complex/clogf.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

#endif /* ! codereview */
/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2005 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __clogf = clogf

#include "libm.h"
#include "complex_wrapper.h"

#if defined(__i386) && !defined(__amd64)
extern int __swapRP(int);
#endif

fcomplex
clogf(fcomplex z)
{
clogf(fcomplex z) {
        fcomplex ans;
        float x, y, ax, ay;
        double dx, dy;
        int ix, iy, hx, hy;

        x = F_RE(z);
        y = F_IM(z);
        hx = THE_WORD(x);
        hy = THE_WORD(y);
        ix = hx & 0x7fffffff;
        iy = hy & 0x7fffffff;
        ay = fabsf(y);
        ax = fabsf(x);
        F_IM(ans) = atan2f(y, x);

#endif /* ! codereview */
```



```
        if (ix >= 0x7f800000 || iy >= 0x7f800000) {
                /* x or y is Inf or NaN */
                if (iy == 0x7f800000)
                        F_RE(ans) = ay;
                else if (ix == 0x7f800000)
                        F_RE(ans) = ax;
                else
                        F_RE(ans) = ax + ay;
        } else {
#if defined(__i386) && !defined(__amd64)
                int rp = __swapRP(fp_extended);
#endif
                dx = (double)ax;
                dy = (double)ay;

#endif /* ! codereview */
                if (ix == 0x3f800000)
                        F_RE(ans) = (float)(0.5 * log1p(dy * dy));
                else if (iy == 0x3f800000)
                        F_RE(ans) = (float)(0.5 * log1p(dx * dx));
                else if ((ix | iy) == 0)
                        F_RE(ans) = -1.0f / ax;
                else
                        F_RE(ans) = (float)(0.5 * log(dx * dx + dy * dy));

#endif /* ! codereview */
#if defined(__i386) && !defined(__amd64)
                if (rp != fp_extended)
                        (void) __swapRP(rp);
#endif
        }

#endif /* ! codereview */
        return (ans);
}
```

```
**********************************************************
    2706 Thu Jun  6 20:39:46 2019
new/usr/src/lib/libm/common/complex/clogl.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __clogl = clogl

#include "libm.h"       /* atan2l/fabsl/isinfl/log1pl/logl/__k_clog_rl */
#include "complex_wrapper.h"
#include "longdouble.h"

#if defined(__sparc)
#define SIGP7           120
#define HSIGP7          60
#elif defined(__x86)
#define SIGP7           70
#define HSIGP7          35
#endif

/* INDENT OFF */
static const long double zero = 0.0L, half = 0.5L, one = 1.0L;

/* INDENT ON */

ldcomplex
clogl(ldcomplex z)
{
clogl(ldcomplex z) {
        ldcomplex ans;
        long double x, y, t, ax, ay;
        int n, ix, iy, hx, hy;

        x = LD_RE(z);
        y = LD_IM(z);
        hx = HI_XWORD(x);
```



```
        hy = HI_XWORD(y);
        ix = hx & 0x7fffffff;
        iy = hy & 0x7fffffff;
        ay = fabsl(y);
        ax = fabsl(x);
        LD_IM(ans) = atan2l(y, x);

#endif /* ! codereview */
        if (ix < iy || (ix == iy && ix < 0x7fff0000 && ax < ay)) {
                /* swap x and y to force ax>=ay */
                t = ax;
                ax = ay;
                ay = t;
                n = ix, ix = iy;
                iy = n;
        }

#endif /* ! codereview */
        n = (ix - iy) >> 16;

#endif /* ! codereview */
        if (ix >= 0x7fff0000) {         /* x or y is Inf or NaN */
                if (isinfl(ax))
                        LD_RE(ans) = ax;
                else if (isinfl(ay))
                        LD_RE(ans) = ay;
                else
                        LD_RE(ans) = ax + ay;
        } else if (ay == zero) {
        } else if (ay == zero)
                LD_RE(ans) = logl(ax);
        } else if (((0x3fffffff - ix) ^ (ix - 0x3ffe0000)) >= 0) {
        else if (((0x3fffffff - ix) ^ (ix - 0x3ffe0000)) >= 0) {
                /* 0.5 <= x < 2 */
                if (ix >= 0x3fff0000) {
                        if (ax == one)
                                LD_RE(ans) = half * log1pl(ay * ay);
                        else if (n >= SIGP7)
                                LD_RE(ans) = logl(ax);
                        else
                                LD_RE(ans) = half * (log1pl(ay * ay + (ax -
                                    one) * (ax + one)));
                } else if (n >= SIGP7) {
                } else if (n >= SIGP7)
                        LD_RE(ans) = logl(ax);
                } else {
                else
                        LD_RE(ans) = __k_clog_rl(x, y, &t);
                }
        } else if (n >= HSIGP7) {
        } else if (n >= HSIGP7)
                LD_RE(ans) = logl(ax);
        } else if (ix < 0x5f3f0000 && iy >= 0x20bf0000) {
        else if (ix < 0x5f3f0000 && iy >= 0x20bf0000)
                /* 2**-8000 < y < x < 2**8000 */
                LD_RE(ans) = half * logl(ax * ax + ay * ay);
        } else {
        else {
                t = ay / ax;
                LD_RE(ans) = logl(ax) + half * log1pl(t * t);
        }

#endif /* ! codereview */
        return (ans);
}
```

```
**********************************************************
    2735 Thu Jun  6 20:39:47 2019
new/usr/src/lib/libm/common/complex/complex_wrapper.h
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#ifndef _COMPLEX_WRAPPER_H
#define _COMPLEX_WRAPPER_H

#pragma ident   "@(#)complex_wrapper.h  1.7     06/01/31 SMI"

#if defined(__GNUC__)
#define dcomplex                double _Complex
#define fcomplex                float _Complex
#define ldcomplex               long double _Complex
#define D_RE(x)                 __real__ x
#define D_IM(x)                 __imag__ x
#define F_RE(x)                 __real__ x
#define F_IM(x)                 __imag__ x
#define LD_RE(x)                __real__ x
#define LD_IM(x)                __imag__ x

#include <complex.h>
#else

#define dcomplex                double complex
#define fcomplex                float complex
#define ldcomplex               long double complex
#define _X_RE(__t, __z)         ((__t *)&__z)[0]
#define _X_IM(__t, __z)         ((__t *)&__z)[1]
#define _X_RE(__t, __z) ((__t *) &__z)[0]
#define _X_IM(__t, __z) ((__t *) &__z)[1]
#define D_RE(__z)               _X_RE(double, __z)
#define D_IM(__z)               _X_IM(double, __z)
#define F_RE(__z)               _X_RE(float, __z)
#define F_IM(__z)               _X_IM(float, __z)
```



```
#define LD_RE(__z)              _X_RE(long double, __z)
#define LD_IM(__z)              _X_IM(long double, __z)

#include <complex.h>
#endif

#if defined(__sparc)
#define HIWORD          0
#define LOWORD          1
#define HI_XWORD(x)     ((unsigned *)&x)[0]
#define XFSCALE(x, n)   ((unsigned *)&x)[0] += n << 16  /* signbitl(x) == 0 */
#define CHOPPED(x)      ((long double)((double)(x)))
#define HI_XWORD(x)     ((unsigned *) &x)[0]
#define XFSCALE(x, n)   ((unsigned *) &x)[0] += n << 16 /* signbitl(x) == 0 */
#define CHOPPED(x)      ((long double) ((double) (x)))
#elif defined(__x86)
#define HIWORD          1
#define LOWORD          0
#define HI_XWORD(x)     ((((int *)&x)[2] << 16) | (0xffff & \
        ((unsigned *)&x)[1] >> 15))
#define XFSCALE(x, n)   ((unsigned short *)&x)[4] += n  /* signbitl(x) == 0 */
#define CHOPPED(x)      ((long double)((float)(x)))
#define HI_XWORD(x)     ((((int *) &x)[2] << 16) | \
                        (0xffff & ((unsigned *) &x)[1] >> 15))
#define XFSCALE(x, n)   ((unsigned short *) &x)[4] += n /* signbitl(x) == 0 */
#define CHOPPED(x)      ((long double) ((float) (x)))
#else
#error Unknown architecture
#endif
#define HI_WORD(x)              ((int *)&x)[HIWORD]     /* for double */
#define LO_WORD(x)              ((int *)&x)[LOWORD]     /* for double */
#define THE_WORD(x)             ((int *)&x)[0]          /* for float */
#define HI_WORD(x)      ((int *) &x)[HIWORD]    /* for double */
#define LO_WORD(x)      ((int *) &x)[LOWORD]    /* for double */
#define THE_WORD(x)     ((int *) &x)[0]         /* for float */

/*
 * iy:ly must have the sign bit already cleared
 */
#define ISINF(iy, ly)           (((iy - 0x7ff00000) | ly) == 0)

#endif /* _COMPLEX_WRAPPER_H */
```

```
**********************************************************
    1138 Thu Jun  6 20:39:47 2019
new/usr/src/lib/libm/common/complex/conj.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __conj = conj

#include "libm.h"
#include "complex_wrapper.h"

dcomplex
conj(dcomplex z)
{
conj(dcomplex z) {
        D_IM(z) = -D_IM(z);
        return (z);
}
______unchanged_portion_omitted_
```

```
**********************************************************
    1141 Thu Jun  6 20:39:48 2019
new/usr/src/lib/libm/common/complex/conjf.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __conjf = conjf

#include "libm.h"
#include "complex_wrapper.h"

fcomplex
conjf(fcomplex z)
{
conjf(fcomplex z) {
        F_IM(z) = -F_IM(z);
        return (z);
}
______unchanged_portion_omitted_
```

```
**********************************************************
    1145 Thu Jun  6 20:39:48 2019
new/usr/src/lib/libm/common/complex/conjl.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __conjl = conjl

#include "libm.h"
#include "complex_wrapper.h"

ldcomplex
conjl(ldcomplex z)
{
conjl(ldcomplex z) {
        LD_IM(z) = -LD_IM(z);
        return (z);
}
______unchanged_portion_omitted_
```

```
**********************************************************
    9572 Thu Jun  6 20:39:49 2019
new/usr/src/lib/libm/common/complex/cpow.c
11210 libm should be cstyle(10NBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __cpow = cpow


/* INDENT OFF */
/*
 * dcomplex cpow(dcomplex z);
 *
 * z**w analytically equivalent to
 *
 * cpow(z,w) = cexp(w clog(z))
 *
 * Let z = x+iy, w = u+iv.
 * Since
 *                        ________
 *                       / 2    2            -1   y
 *     log(x+iy) = log(\/ x  + y    ) + i tan   (---)
 *                                              x
 *
 *                  1       2    2         -1   y
 *               = --- log(x  + y ) + i tan   (---)
 *                  2                       x
 *                       u       2    2         -1  y
 * (u+iv)* log(x+iy) =  --- log(x  + y ) - v tan  (---)  +          (1)
 *                       2                         x
 *
 *                           v       2    2         -1  y
 *                     i * [ --- log(x  + y ) + u tan  (---) ]      (2)
 *                           2                         x
 *
 *                   = r + i q
```



```
 *
 * Therefore,
 *      w     r+iq    r
 *     z  =  e     = e  (cos(q)+i*sin(q))
 *                                   ________
 *                                  / 2   2
 *       r                        \/ x + y     -v*atan2(y,x)
 * Here e  can be expressed as:  u          * e
 *
 * Special cases (in the order of appearance):
 *      1.  (anything) ** 0  is 1
 *      2.  (anything) ** 1  is itself
 *      3.  When v = 0, y = 0:
 *            If x is finite and negative, and u is finite, then
 *               x ** u = exp(u*pi i) * pow(|x|, u);
 *            otherwise,
 *               x ** u = pow(x, u);
 *      4.  When v = 0, x = 0 or |x| = |y| or x is inf or y is inf:
 *               (x + y i) ** u = r * exp(q i)
 *          where
 *               r = hypot(x,y) ** u
 *               q = u * atan2pi(y, x)
 *
 *      5.  otherwise, z**w is NAN if any x, y, u, v is a Nan or inf
 *
 *      Note: many results of special cases are obtained in terms of
 *      polar coordinate. In the conversion from polar to rectangle:
 *                  r exp(q i) = r * cos(q) + r * sin(q) i,
 *      we regard r * 0 is 0 except when r is a NaN.
 */
/* INDENT ON */

/*
 * atan2/exp/fabs/hypot/log/pow/scalbn
 * atan2pi/exp2/sincos/sincospi/__k_clog_r/__k_atan2
 */
#include "libm.h"
#include "libm.h"       /* atan2/exp/fabs/hypot/log/pow/scalbn */
                        /* atan2pi/exp2/sincos/sincospi/__k_clog_r/__k_atan2 */
#include "complex_wrapper.h"

extern void sincospi(double, double *, double *);
static const double huge = 1e300,

static const double
        huge = 1e300,
        tiny = 1e-300,
        invln2 = 1.4426950408896338700e+00,
        ln2hi = 6.9314718036912381649 0e-01,     /* 0x3fe62e42, 0xfee00000 */
        ln2lo = 1.9082149292705877000 2e-10,     /* 0x3dea39ef, 0x35793c76 */
        one = 1.0,
        zero = 0.0;

static const int hiinf = 0x7ff00000;
extern double atan2pi(double, double);

/*
 * Assuming |t[0]| > |t[1]| and |t[2]| > |t[3]|, sum4fp subroutine
 * compute t[0] + t[1] + t[2] + t[3] into two double fp numbers.
 */
static double
sum4fp(double ta[], double *w)
{
sum4fp(double ta[], double *w) {
        double t1, t2, t3, t4, w1, w2, t;
```

```
119         t1 = ta[0];
120         t2 = ta[1];
121         t3 = ta[2];
122         t4 = ta[3];

109         t1 = ta[0]; t2 = ta[1]; t3 = ta[2]; t4 = ta[3];
124         /*
125          * Rearrange ti so that |t1| >= |t2| >= |t3| >= |t4|
126          */
127         if (fabs(t4) > fabs(t1)) {
128                 t = t1;
129                 t1 = t3;
130                 t3 = t;
131                 t = t2;
132                 t2 = t4;
133                 t4 = t;
114                 t = t1; t1 = t3; t3 = t;
115                 t = t2; t2 = t4; t4 = t;
134         } else if (fabs(t3) > fabs(t1)) {
135                 t = t1;
136                 t1 = t3;

117                 t = t1; t1 = t3;
138                 if (fabs(t4) > fabs(t2)) {
139                         t3 = t4;
140                         t4 = t2;
141                         t2 = t;
119                         t3 = t4; t4 = t2; t2 = t;
142                 } else {
143                         t3 = t2;
144                         t2 = t;
121                         t3 = t2; t2 = t;
145                 }
146         } else if (fabs(t3) > fabs(t2)) {
147                 t = t2;
148                 t2 = t3;

124                 t = t2; t2 = t3;
150                 if (fabs(t4) > fabs(t2)) {
151                         t3 = t4;
152                         t4 = t;
153                 } else {
126                         t3 = t4; t4 = t;
127                 } else
154                         t3 = t;
155                 }
156         }

158 #endif /* ! codereview */
159         /* summing r = t1 + t2 + t3 + t4 to w1 + w2 */
160         w1 = t3 + t4;
161         w2 = t4 - (w1 - t3);
162         t = t2 + w1;
163         w2 += w1 - (t - t2);
164         w1 = t + w2;
165         w2 += t - w1;
166         t = t1 + w1;
167         w2 += w1 - (t - t1);
168         w1 = t + w2;
169         *w = w2 - (w1 - t);
170         return (w1);
171 }

173 dcomplex
174 cpow(dcomplex z, dcomplex w)
175 {
```



```
130 cpow(dcomplex z, dcomplex w) {
176         dcomplex ans;
177         double x, y, u, v, t, c, s, r, x2, y2;
178         double b[4], t1, t2, t3, t4, w1, w2, u1, v1, x1, y1;
179         int ix, iy, hx, lx, hy, ly, hv, hu, iu, iv, lu, lv;
180         int i, j, k;

182         x = D_RE(z);
183         y = D_IM(z);
184         u = D_RE(w);
185         v = D_IM(w);
186         hx = ((int *)&x)[HIWORD];
187         lx = ((int *)&x)[LOWORD];
188         hy = ((int *)&y)[HIWORD];
189         ly = ((int *)&y)[LOWORD];
190         hu = ((int *)&u)[HIWORD];
191         lu = ((int *)&u)[LOWORD];
192         hv = ((int *)&v)[HIWORD];
193         lv = ((int *)&v)[LOWORD];
141         hx = ((int *) &x)[HIWORD];
142         lx = ((int *) &x)[LOWORD];
143         hy = ((int *) &y)[HIWORD];
144         ly = ((int *) &y)[LOWORD];
145         hu = ((int *) &u)[HIWORD];
146         lu = ((int *) &u)[LOWORD];
147         hv = ((int *) &v)[HIWORD];
148         lv = ((int *) &v)[LOWORD];
194         ix = hx & 0x7fffffff;
195         iy = hy & 0x7fffffff;
196         iu = hu & 0x7fffffff;
197         iv = hv & 0x7fffffff;

199         j = 0;

201 #endif /* ! codereview */
202         if ((iv | lv) == 0) {                           /* z**(real) */
203                 if (((hu - 0x3ff00000) | lu) == 0) {    /* z ** 1 = z */
204                         D_RE(ans) = x;
205                         D_IM(ans) = y;
206                 } else if ((iu | lu) == 0) {            /* z ** 0 = 1 */
207                         D_RE(ans) = one;
208                         D_IM(ans) = zero;
209                 } else if ((iy | ly) == 0) {            /* (real)**(real) */
210                         D_IM(ans) = zero;

212 #endif /* ! codereview */
213                         if (hx < 0 && ix < hiinf && iu < hiinf) {
214                                 /* -x ** u  is exp(i*pi*u)*pow(x,u) */
215                                 r = pow(-x, u);
216                                 sincospi(u, &s, &c);
217                                 D_RE(ans) = (c == zero) ? c : c *r;
218                                 D_IM(ans) = (s == zero) ? s : s *r;
219                         } else {
155                                 D_RE(ans) = (c == zero)? c: c * r;
156                                 D_IM(ans) = (s == zero)? s: s * r;
157                         } else
220                                 D_RE(ans) = pow(x, u);
221                         }
222 #endif /* ! codereview */
223                 } else if (((ix | lx) == 0) || ix >= hiinf || iy >= hiinf) {
224                         if (isnan(x) || isnan(y) || isnan(u)) {
159                         if (isnan(x) || isnan(y) || isnan(u))
225                                 D_RE(ans) = D_IM(ans) = x + y + u;
226                         } else {
161                         else {
227                                 if ((ix | lx) == 0)
```

```
228                                     r = fabs(y);
229                             else
230                                     r = fabs(x) + fabs(y);

232 #endif /* ! codereview */
233                             t = atan2pi(y, x);
234                             sincospi(t * u, &s, &c);
235                             D_RE(ans) = (c == zero) ? c : c *r;
236                             D_IM(ans) = (s == zero) ? s : s *r;
166                             D_RE(ans) = (c == zero)? c: c * r;
167                             D_IM(ans) = (s == zero)? s: s * r;
237                     }
238             } else if (((ix - iy) | (lx - ly)) == 0) {      /* |x| = |y| */
239                     if (hx >= 0) {
240                             t = (hy >= 0) ? 0.25 : -0.25;
171                             t = (hy >= 0)? 0.25 : -0.25;
241                             sincospi(t * u, &s, &c);
242                     } else if ((lu & 3) == 0) {
243                             t = (hy >= 0) ? 0.75 : -0.75;
174                             t = (hy >= 0)? 0.75 : -0.75;
244                             sincospi(t * u, &s, &c);
245                     } else {
246                             r = (hy >= 0) ? u : -u;
177                             r = (hy >= 0)? u : -u;
247                             t = -0.25 * r;
248                             w1 = r + t;
249                             w2 = t - (w1 - r);
250                             sincospi(w1, &t1, &t2);
251                             sincospi(w2, &t3, &t4);
252                             s = t1 * t4 + t3 * t2;
253                             c = t2 * t4 - t1 * t3;
254                     }

256 #endif /* ! codereview */
257                     if (ix < 0x3fe00000)    /* |x| < 1/2 */
258                             r = pow(fabs(x + x), u) * exp2(-0.5 * u);
259                     else if (ix >= 0x3ff00000 || iu < 0x408ff800)
260                             /* |x| >= 1 or |u| < 1023 */
261                             r = pow(fabs(x), u) * exp2(0.5 * u);
262                     else            /* special treatment */
263                             j = 2;

265 #endif /* ! codereview */
266                     if (j == 0) {
267                             D_RE(ans) = (c == zero) ? c : c *r;
268                             D_IM(ans) = (s == zero) ? s : s *r;
186                             D_RE(ans) = (c == zero)? c: c * r;
187                             D_IM(ans) = (s == zero)? s: s * r;
269                     }
270             } else {
189             } else
271                     j = 1;
272             }

274 #endif /* ! codereview */
275             if (j == 0)
276                     return (ans);
277     }

279 #endif /* ! codereview */
280     if (iu >= hiinf || iv >= hiinf || ix >= hiinf || iy >= hiinf) {
281             /*
282              * non-zero imag part(s) with inf component(s) yields NaN
283              */
284             t = fabs(x) + fabs(y) + fabs(u) + fabs(v);
285             D_RE(ans) = D_IM(ans) = t - t;
```



```
286     } else {
287             k = 0;                          /* no scaling */

289 #endif /* ! codereview */
290             if (iu > 0x7f000000 || iv > 0x7f000000) {
291                     u *= .0009765625; /* scale 2**-10 to avoid overflow */
292                     v *= .0009765625;
293                     k = 1;                  /* scale by 2**-10 */
294             }

296 #endif /* ! codereview */
297             /*
298              * Use similated higher precision arithmetic to compute:
299              * r = u * log(hypot(x, y)) - v * atan2(y, x)
300              * q = u * atan2(y, x) + v * log(hypot(x, y))
301              */
302             t1 = __k_clog_r(x, y, &t2);
303             t3 = __k_atan2(y, x, &t4);
304             x1 = t1;
305             y1 = t3;
306             u1 = u;
307             v1 = v;
308             ((int *)&u1)[LOWORD] &= 0xf8000000;
309             ((int *)&v1)[LOWORD] &= 0xf8000000;
310             ((int *)&x1)[LOWORD] &= 0xf8000000;
311             ((int *)&y1)[LOWORD] &= 0xf8000000;
191             ((int *) &u1)[LOWORD] &= 0xf8000000;
192             ((int *) &v1)[LOWORD] &= 0xf8000000;
193             ((int *) &x1)[LOWORD] &= 0xf8000000;
194             ((int *) &y1)[LOWORD] &= 0xf8000000;
312             x2 = t2 - (x1 - t1);    /* log(hypot(x,y)) = x1 + x2 */
313             y2 = t4 - (y1 - t3);    /* atan2(y,x) = y1 + y2 */

315 #endif /* ! codereview */
316             /* compute q = u * atan2(y, x) + v * log(hypot(x, y)) */
317             if (j != 2) {
318                     b[0] = u1 * y1;
319                     b[1] = (u - u1) * y1 + u * y2;

321 #endif /* ! codereview */
322                     if (j == 1) {   /* v = 0 */
323                             w1 = b[0] + b[1];
324                             w2 = b[1] - (w1 - b[0]);
325                     } else {
326                             b[2] = v1 * x1;
327                             b[3] = (v - v1) * x1 + v * x2;
328                             w1 = sum4fp(b, &w2);
329                     }

331 #endif /* ! codereview */
332                     sincos(w1, &t1, &t2);
333                     sincos(w2, &t3, &t4);
334                     s = t1 * t4 + t3 * t2;
335                     c = t2 * t4 - t1 * t3;

337                     if (k == 1) {
197                     if (k == 1)
338                             /*
339                              * square (cos(q) + i sin(q)) k times to get
340                              * (cos(2^k * q + i sin(2^k * q)
341                              */
342                             for (i = 0; i < 10; i++) {
343                                     t1 = s * c;
344                                     c = (c + s) * (c - s);
345                                     s = t1 + t1;
346                             }
```

```
347                         }
348                 }

350 #endif /* ! codereview */
351                 /* compute r = u * (t1, t2) - v * (t3, t4) */
352                 b[0] = u1 * x1;
353                 b[1] = (u - u1) * x1 + u * x2;

355 #endif /* ! codereview */
356                 if (j == 1) {           /* v = 0 */
357                         w1 = b[0] + b[1];
358                         w2 = b[1] - (w1 - b[0]);
359                 } else {
360                         b[2] = -v1 * y1;
361                         b[3] = (v1 - v) * y1 - v * y2;
362                         w1 = sum4fp(b, &w2);
363                 }

365 #endif /* ! codereview */
366                 /* check over/underflow for exp(w1 + w2) */
367                 if (k && fabs(w1) < 1000.0) {
368                         w1 *= 1024;
369                         w2 *= 1024;
370                         k = 0;
208                         w1 *= 1024; w2 *= 1024; k = 0;
371                 }

373                 hx = ((int *)&w1)[HIWORD];
374                 lx = ((int *)&w1)[LOWORD];
210                 hx = ((int *) &w1)[HIWORD];
211                 lx = ((int *) &w1)[LOWORD];
375                 ix = hx & 0x7fffffff;

377 #endif /* ! codereview */
378                 /* compute exp(w1 + w2) */
379                 if (ix < 0x3c900000) {          /* exp(tiny < 2**-54) = 1 */
213                 if (ix < 0x3c900000) /* exp(tiny < 2**-54) = 1 */
380                         r = one;
381                 } else if (ix >= 0x40880000) {  /* overflow/underflow */
382                         r = (hx < 0) ? tiny * tiny : huge * huge;
383                 } else { /* compute exp(w1 + w2) */
384                         k = (int)(invln2 * w1 + ((hx >= 0) ? 0.5 : -0.5));
385                         t1 = (double)k;
215                 else if (ix >= 0x40880000) /* overflow/underflow */
216                         r = (hx < 0)? tiny * tiny : huge * huge;
217                 else {  /* compute exp(w1 + w2) */
218                         k = (int) (invln2 * w1 + ((hx >= 0)? 0.5 : -0.5));
219                         t1 = (double) k;
386                         t2 = w1 - t1 * ln2hi;
387                         t3 = w2 - t1 * ln2lo;
388                         r = exp(t2 + t3);
389                 }

391                 if (c != zero)
392                         c *= r;

394                 if (s != zero)
395                         s *= r;

224                 if (c != zero) c *= r;
225                 if (s != zero) s *= r;
397                 if (k != 0) {
398                         c = scalbn(c, k);
399                         s = scalbn(s, k);
400                 }
```



```
402 #endif /* ! codereview */
403                 D_RE(ans) = c;
404                 D_IM(ans) = s;
405         }

407 #endif /* ! codereview */
408         return (ans);
409 }
```

```
**********************************************************
    4753 Thu Jun  6 20:39:49 2019
new/usr/src/lib/libm/common/complex/cpowf.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

#endif /* ! codereview */
/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2005 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __cpowf = cpowf

#include "libm.h"
#include "complex_wrapper.h"

extern void sincospi(double, double *, double *);
extern void sincospif(float, float *, float *);
extern double atan2pi(double, double);
extern float atan2pif(float, float);

#if defined(__i386) && !defined(__amd64)
extern int __swapRP(int);
#endif

/* Hex 2^ 1 * 1.921FB54442D18 */
static const double dpi = 3.1415926535897931160E0;

static const double dhalf = 0.5,
static const double
        dpi = 3.1415926535897931160E0,  /* Hex 2^ 1 * 1.921FB54442D18 */
        dhalf = 0.5,
        dsqrt2 = 1.4142135623730951454 7,        /* 3FF6A09E 667F3BCD */
        dinvpi = 0.31830988618379067153 77675;

static const float one = 1.0F, zero = 0.0F;

#define hiinf   0x7f800000

fcomplex
```



```
cpowf(fcomplex z, fcomplex w)
{
cpowf(fcomplex z, fcomplex w) {
        fcomplex ans;
        float x, y, u, v, t, c, s;
        double dx, dy, du, dv, dt, dc, ds, dp, dq, dr;
        int ix, iy, hx, hy, hv, hu, iu, iv, j;

        x = F_RE(z);
        y = F_IM(z);
        u = F_RE(w);
        v = F_IM(w);
        hx = THE_WORD(x);
        hy = THE_WORD(y);
        hu = THE_WORD(u);
        hv = THE_WORD(v);
        ix = hx & 0x7fffffff;
        iy = hy & 0x7fffffff;
        iu = hu & 0x7fffffff;
        iv = hv & 0x7fffffff;

        j = 0;

#endif /* ! codereview */
        if (iv == 0) {                  /* z**(real) */
                if (hu == 0x3f800000) { /* (anything) ** 1  is itself */
                        F_RE(ans) = x;
                        F_IM(ans) = y;
                } else if (iu == 0) {   /* (anything) ** 0  is 1 */
                        F_RE(ans) = one;
                        F_IM(ans) = zero;
                } else if (iy == 0) {   /* (real)**(real) */
                        F_IM(ans) = zero;

#endif /* ! codereview */
                        if (hx < 0 && ix < hiinf && iu < hiinf) {
                                /* -x ** u  is exp(i*pi*u)*pow(x,u) */
                                t = powf(-x, u);
                                sincospif(u, &s, &c);
                                F_RE(ans) = (c == zero) ? c : c *t;
                                F_IM(ans) = (s == zero) ? s : s *t;
                                F_RE(ans) = (c == zero)? c: c * t;
                                F_IM(ans) = (s == zero)? s: s * t;
                        } else {
                                F_RE(ans) = powf(x, u);
                        }
                } else if (ix == 0 || ix >= hiinf || iy >= hiinf) {
                        if (ix > hiinf || iy > hiinf || iu > hiinf) {
                                F_RE(ans) = F_IM(ans) = x + y + u;
                        } else {
                                v = fabsf(y);

#endif /* ! codereview */
                                if (ix != 0)
                                        v += fabsf(x);

#endif /* ! codereview */
                                t = atan2pif(y, x);
                                sincospif(t * u, &s, &c);
                                F_RE(ans) = (c == zero) ? c : c *v;
                                F_IM(ans) = (s == zero) ? s : s *v;
                                F_RE(ans) = (c == zero)? c: c * v;
                                F_IM(ans) = (s == zero)? s: s * v;
                        }
                } else if (ix == iy) {  /* if |x| == |y| */
#if defined(__i386) && !defined(__amd64)
```

```
                        int rp = __swapRP(fp_extended);
#endif
                        dx = (double)x;
                        du = (double)u;
                        dt = (hx >= 0) ? 0.25 : 0.75;

                        dt = (hx >= 0)? 0.25 : 0.75;
                        if (hy < 0)
                                dt = -dt;

#endif /* ! codereview */
                        dr = pow(dsqrt2 * dx, du);
                        sincospi(dt * du, &ds, &dc);
                        F_RE(ans) = (float)(dr * dc);
                        F_IM(ans) = (float)(dr * ds);
#if defined(__i386) && !defined(__amd64)
                        if (rp != fp_extended)
                                (void) __swapRP(rp);
#endif
                } else {
                        j = 1;
                }

#endif /* ! codereview */
                if (j == 0)
                        return (ans);
        }

#endif /* ! codereview */
        if (iu >= hiinf || iv >= hiinf || ix >= hiinf || iy >= hiinf) {
                /*
                 * non-zero imaginery part(s) with inf component(s) yields NaN
                 */
                t = fabsf(x) + fabsf(y) + fabsf(u) + fabsf(v);
                F_RE(ans) = F_IM(ans) = t - t;
        } else {
#if defined(__i386) && !defined(__amd64)
                int rp = __swapRP(fp_extended);
#endif

                /* INDENT OFF */
                /*
                 * r = u*log(hypot(x,y))-v*atan2(y,x),
                 * q = u*atan2(y,x)+v*log(hypot(x,y))
                 * or
                 * r = u*log(hypot(x,y))-v*pi*atan2pi(y,x),
                 * q/pi = u*atan2pi(y,x)+v*log(hypot(x,y))/pi
                 * ans = exp(r)*(cospi(q/pi)  + i sinpi(q/pi))
                 */
                /* INDENT ON */
                dx = (double)x;
                dy = (double)y;
                du = (double)u;
                dv = (double)v;

#endif /* ! codereview */
                if (ix > 0x3f000000 && ix < 0x40000000) /* .5 < |x| < 2 */
                        dt = dhalf * log1p((dx - 1.0) * (dx + 1.0) + dy * dy);
                else if (iy > 0x3f000000 && iy < 0x40000000) /* .5 < |y| < 2 */
                        dt = dhalf * log1p((dy - 1.0) * (dy + 1.0) + dx * dx);
                else
                        dt = dhalf * log(dx * dx + dy * dy);

#endif /* ! codereview */
                dp = atan2pi(dy, dx);
```



```
#endif /* ! codereview */
                if (iv == 0) {          /* dv = 0 */
                        dr = exp(du * dt);
                        dq = du * dp;
                } else {
                        dr = exp(du * dt - dv * dp * dpi);
                        dq = du * dp + dv * dt * dinvpi;
                }

#endif /* ! codereview */
                sincospi(dq, &ds, &dc);
                F_RE(ans) = (float)(dr * dc);
                F_IM(ans) = (float)(dr * ds);
#if defined(__i386) && !defined(__amd64)
                if (rp != fp_extended)
                        (void) __swapRP(rp);
#endif
        }

#endif /* ! codereview */
        return (ans);
}
```

```
**********************************************************
    7723 Thu Jun  6 20:39:50 2019
new/usr/src/lib/libm/common/complex/cpowl.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __cpowl = cpowl

#include "libm.h"                       /* __k_clog_rl/__k_atan2l */
/* atan2l/atan2pil/exp2l/expl/fabsl/hypotl/isinfl/logl/powl/sincosl/sincospil */
#include "complex_wrapper.h"
#include "longdouble.h"

#if defined(__sparc)
#define HALF(x)         ((int *)&x)[3] = 0; ((int *)&x)[2] &= 0xfe000000
#define LAST(x)         ((int *)&x)[3]
#define HALF(x)  ((int *) &x)[3] = 0; ((int *) &x)[2] &= 0xfe000000
#define LAST(x)  ((int *) &x)[3]
#elif defined(__x86)
#define HALF(x)         ((int *)&x)[0] = 0
#define LAST(x)         ((int *)&x)[0]
#define HALF(x)  ((int *) &x)[0] = 0
#define LAST(x)  ((int *) &x)[0]
#endif

/* INDENT OFF */
static const int hiinf = 0x7fff0000;
static const long double tiny = 1.0e-4000L,
static const long double
        tiny = 1.0e-4000L,
        huge = 1.0e4000L,
#if defined(__x86)
/* 43 significant bits, 21 trailing zeros */
                /* 43 significant bits, 21 trailing zeros */
        ln2hil = 0.693147180559890330187045037746429443359375L,
        ln2lol = 5.497923018708371174712471612513436025525412068e-14L,
```



```
#else /* sparc */
/* 0x3FF962E4 2FEFA39E F35793C7 00000000 */
                /* 0x3FF962E4 2FEFA39E F35793C7 00000000 */
        ln2hil = 0.693147180559945309417231592858066493070671489074L,
        ln2lol = 5.286001100750048286452862358206467301068024465661 53e-25L,
#endif
        invln2 = 1.442695040888963407359924681001892137427e+0000L,
        one = 1.0L,
        zero = 0.0L;
/* INDENT ON */

/*
 * Assuming |t[0]| > |t[1]| and |t[2]| > |t[3]|, sum4fpl subroutine
 * compute t[0] + t[1] + t[2] + t[3] into two long double fp numbers.
 */
static long double
sum4fpl(long double ta[], long double *w)
static long double sum4fpl(long double ta[], long double *w)
{
        long double t1, t2, t3, t4, w1, w2, t;

        t1 = ta[0];
        t2 = ta[1];
        t3 = ta[2];
        t4 = ta[3];

        t1 = ta[0]; t2 = ta[1]; t3 = ta[2]; t4 = ta[3];
        /*
         * Rearrange ti so that |t1| >= |t2| >= |t3| >= |t4|
         */
        if (fabsl(t4) > fabsl(t1)) {
                t = t1;
                t1 = t3;
                t3 = t;
                t = t2;
                t2 = t4;
                t4 = t;
                t = t1; t1 = t3; t3 = t;
                t = t2; t2 = t4; t4 = t;
        } else if (fabsl(t3) > fabsl(t1)) {
                t = t1;
                t1 = t3;

                t = t1; t1 = t3;
                if (fabsl(t4) > fabsl(t2)) {
                        t3 = t4;
                        t4 = t2;
                        t2 = t;
                        t3 = t4; t4 = t2; t2 = t;
                } else {
                        t3 = t2;
                        t2 = t;
                        t3 = t2; t2 = t;
                }
        } else if (fabsl(t3) > fabsl(t2)) {
                t = t2;
                t2 = t3;

                t = t2; t2 = t3;
                if (fabsl(t4) > fabsl(t2)) {
                        t3 = t4;
                        t4 = t;
                } else {
                        t3 = t4; t4 = t;
                } else
                        t3 = t;
```

```
                }
        }

#endif /* ! codereview */
        /* summing r = t1 + t2 + t3 + t4 to w1 + w2 */
        w1 = t3 + t4;
        w2 = t4 - (w1 - t3);
        t = t2 + w1;
        w2 += w1 - (t - t2);
        w1 = t + w2;
        w2 += t - w1;
        t = t1 + w1;
        w2 += w1 - (t - t1);
        w1 = t + w2;
        *w = w2 - (w1 - t);
        return (w1);
}

ldcomplex
cpowl(ldcomplex z, ldcomplex w)
{
cpowl(ldcomplex z, ldcomplex w) {
        ldcomplex ans;
        long double x, y, u, v, t, c, s, r;
        long double t1, t2, t3, t4, x1, x2, y1, y2, u1, v1, b[4], w1, w2;
        int ix, iy, hx, hy, hv, hu, iu, iv, i, j, k;

        x = LD_RE(z);
        y = LD_IM(z);
        u = LD_RE(w);
        v = LD_IM(w);
        hx = HI_XWORD(x);
        hy = HI_XWORD(y);
        hu = HI_XWORD(u);
        hv = HI_XWORD(v);
        ix = hx & 0x7fffffff;
        iy = hy & 0x7fffffff;
        iu = hu & 0x7fffffff;
        iv = hv & 0x7fffffff;

        j = 0;

#endif /* ! codereview */
        if (v == zero) {                /* z**(real) */
                if (u == one) {         /* (anything) ** 1  is itself */
                        LD_RE(ans) = x;
                        LD_IM(ans) = y;
                } else if (u == zero) { /* (anything) ** 0  is 1 */
                        LD_RE(ans) = one;
                        LD_IM(ans) = zero;
                } else if (y == zero) { /* real ** real */
                        LD_IM(ans) = zero;

#endif /* ! codereview */
                        if (hx < 0 && ix < hiinf && iu < hiinf) {
                                /* -x ** u  is exp(i*pi*u)*pow(x,u) */
                                r = powl(-x, u);
                                sincospil(u, &s, &c);
                                LD_RE(ans) = (c == zero) ? c : c *r;
                                LD_IM(ans) = (s == zero) ? s : s *r;
                        } else {
                                LD_RE(ans) = (c == zero)? c: c * r;
                                LD_IM(ans) = (s == zero)? s: s * r;
                        } else
                                LD_RE(ans) = powl(x, u);
                        }
```



```
#endif /* ! codereview */
                } else if (x == zero || ix >= hiinf || iy >= hiinf) {
                        if (isnanl(x) || isnanl(y) || isnanl(u)) {
                        if (isnanl(x) || isnanl(y) || isnanl(u))
                                LD_RE(ans) = LD_IM(ans) = x + y + u;
                        } else {
                        else {
                                if (x == zero)
                                        r = fabsl(y);
                                else
                                        r = fabsl(x) + fabsl(y);

#endif /* ! codereview */
                                t = atan2pil(y, x);
                                sincospil(t * u, &s, &c);
                                LD_RE(ans) = (c == zero) ? c : c *r;
                                LD_IM(ans) = (s == zero) ? s : s *r;
                                LD_RE(ans) = (c == zero)? c: c * r;
                                LD_IM(ans) = (s == zero)? s: s * r;
                        }
                } else if (fabsl(x) == fabsl(y)) {      /* |x| = |y| */
                        if (hx >= 0) {
                                t = (hy >= 0) ? 0.25L : -0.25L;
                                t = (hy >= 0)? 0.25L : -0.25L;
                                sincospil(t * u, &s, &c);
                        } else if ((LAST(u) & 3) == 0) {
                                t = (hy >= 0) ? 0.75L : -0.75L;
                                t = (hy >= 0)? 0.75L : -0.75L;
                                sincospil(t * u, &s, &c);
                        } else {
                                r = (hy >= 0) ? u : -u;
                                r = (hy >= 0)? u : -u;
                                t = -0.25L * r;
                                w1 = r + t;
                                w2 = t - (w1 - r);
                                sincospil(w1, &t1, &t2);
                                sincospil(w2, &t3, &t4);
                                s = t1 * t4 + t3 * t2;
                                c = t2 * t4 - t1 * t3;
                        }

#endif /* ! codereview */
                        if (ix < 0x3ffe0000)    /* |x| < 1/2 */
                                r = powl(fabsl(x + x), u) * exp2l(-0.5L * u);
                        else if (ix >= 0x3fff0000 || iu < 0x400cfff8)
                                /* |x| >= 1 or |u| < 16383 */
                                r = powl(fabsl(x), u) * exp2l(0.5L * u);
                        else            /* special treatment */
                                j = 2;

#endif /* ! codereview */
                        if (j == 0) {
                                LD_RE(ans) = (c == zero) ? c : c *r;
                                LD_IM(ans) = (s == zero) ? s : s *r;
                                LD_RE(ans) = (c == zero)? c: c * r;
                                LD_IM(ans) = (s == zero)? s: s * r;
                        }
                } else {
                } else
                        j = 1;
                }

#endif /* ! codereview */
                if (j == 0)
                        return (ans);
        }
```

```
#endif /* ! codereview */
        if (iu >= hiinf || iv >= hiinf || ix >= hiinf || iy >= hiinf) {
                /*
                 * non-zero imag part(s) with inf component(s) yields NaN
                 */
                t = fabsl(x) + fabsl(y) + fabsl(u) + fabsl(v);
                LD_RE(ans) = LD_IM(ans) = t - t;
        } else {
                k = 0;                  /* no scaling */

#endif /* ! codereview */
                if (iu > 0x7ffe0000 || iv > 0x7ffe0000) {
                        u *= 1.52587890625000000000e-05L;
                        v *= 1.52587890625000000000e-05L;
                        k = 1;          /* scale u and v by 2**-16 */
                }

#endif /* ! codereview */
                /*
                 * Use similated higher precision arithmetic to compute:
                 * r = u * log(hypot(x, y)) - v * atan2(y, x)
                 * q = u * atan2(y, x) + v * log(hypot(x, y))
                 */

                t1 = __k_clog_rl(x, y, &t2);
                t3 = __k_atan2l(y, x, &t4);
                x1 = t1;
                HALF(x1);
                y1 = t3;
                HALF(y1);
                u1 = u;
                HALF(u1);
                v1 = v;
                HALF(v1);
                x1 = t1; HALF(x1);
                y1 = t3; HALF(y1);
                u1 = u; HALF(u1);
                v1 = v; HALF(v1);
                x2 = t2 - (x1 - t1);    /* log(hypot(x,y)) = x1 + x2 */
                y2 = t4 - (y1 - t3);    /* atan2(y,x) = y1 + y2 */

#endif /* ! codereview */
                /* compute q = u * atan2(y, x) + v * log(hypot(x, y)) */
                if (j != 2) {
                        b[0] = u1 * y1;
                        b[1] = (u - u1) * y1 + u * y2;

#endif /* ! codereview */
                        if (j == 1) {   /* v = 0 */
                                w1 = b[0] + b[1];
                                w2 = b[1] - (w1 - b[0]);
                        } else {
                                b[2] = v1 * x1;
                                b[3] = (v - v1) * x1 + v * x2;
                                w1 = sum4fpl(b, &w2);
                        }

#endif /* ! codereview */
                        sincosl(w1, &t1, &t2);
                        sincosl(w2, &t3, &t4);
                        s = t1 * t4 + t3 * t2;
                        c = t2 * t4 - t1 * t3;

                        if (k == 1) {   /* square j times */
                        if (k == 1)     /* square j times */
```



```
                                for (i = 0; i < 10; i++) {
                                        t1 = s * c;
                                        c = (c + s) * (c - s);
                                        s = t1 + t1;
                                }
                        }
                }

#endif /* ! codereview */
                /* compute r = u * (t1, t2) - v * (t3, t4) */
                b[0] = u1 * x1;
                b[1] = (u - u1) * x1 + u * x2;

#endif /* ! codereview */
                if (j == 1) {           /* v = 0 */
                        w1 = b[0] + b[1];
                        w2 = b[1] - (w1 - b[0]);
                } else {
                        b[2] = -v1 * y1;
                        b[3] = (v1 - v) * y1 - v * y2;
                        w1 = sum4fpl(b, &w2);
                }

#endif /* ! codereview */
                /* scale back unless w1 is large enough to cause exception */
                if (k != 0 && fabsl(w1) < 20000.0L) {
                        w1 *= 65536.0L;
                        w2 *= 65536.0L;
                        w1 *= 65536.0L; w2 *= 65536.0L;
                }

#endif /* ! codereview */
                hx = HI_XWORD(w1);
                ix = hx & 0x7fffffff;
                /* compute exp(w1 + w2) */
                k = 0;

                if (ix < 0x3f8c0000) {          /* exp(tiny < 2**-115) = 1 */
                if (ix < 0x3f8c0000) /* exp(tiny < 2**-115) = 1 */
                        r = one;
                } else if (ix >= 0x400c6760) {  /* overflow/underflow */
                        r = (hx < 0) ? tiny * tiny : huge * huge;
                } else { /* compute exp(w1 + w2) */
                        k = (int)(invln2 * w1 + ((hx >= 0) ? 0.5L : -0.5L));
                        t1 = (long double)k;
                else if (ix >= 0x400c6760) /* overflow/underflow */
                        r = (hx < 0)? tiny * tiny : huge * huge;
                else {  /* compute exp(w1 + w2) */
                        k = (int) (invln2 * w1 + ((hx >= 0)? 0.5L : -0.5L));
                        t1 = (long double) k;
                        t2 = w1 - t1 * ln2hil;
                        t3 = w2 - t1 * ln2lol;
                        r = expl(t2 + t3);
                }

                if (c != zero)
                        c *= r;

                if (s != zero)
                        s *= r;

                if (c != zero) c *= r;
                if (s != zero) s *= r;
                if (k != 0) {
                        c = scalbnl(c, k);
                        s = scalbnl(s, k);
```

```
                }

#endif /* ! codereview */
                LD_RE(ans) = c;
                LD_IM(ans) = s;
        }

#endif /* ! codereview */
        return (ans);
}
```

```
**********************************************************
    1692 Thu Jun  6 20:39:50 2019
new/usr/src/lib/libm/common/complex/cproj.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __cproj = cproj


/* INDENT OFF */
/*
 * dcomplex cproj(dcomplex z);
 *
 * If one of the component of z = (x,y) is an inf, then
 *      cproj(z) = (+inf, copysign(0,y));
 * otherwise,
 *      cproj(z) = z
 */
/* INDENT ON */

#include "libm.h"                       /* fabs */
#include "complex_wrapper.h"

static const double zero = 0.0;

dcomplex
cproj(dcomplex z)
{
cproj(dcomplex z) {
        double x, y;
        int ix, iy, hx, hy, lx, ly;

        x = D_RE(z);
        y = D_IM(z);
        hx = HI_WORD(x);
        lx = LO_WORD(x);
```



```
        hy = HI_WORD(y);
        ly = LO_WORD(y);
        ix = hx & 0x7fffffff;
        iy = hy & 0x7fffffff;

#endif /* ! codereview */
        if (ISINF(iy, ly)) {
                D_RE(z) = fabs(y);
                D_IM(z) = hy >= 0 ? zero : -zero;
        } else if (ISINF(ix, lx)) {
                D_RE(z) = fabs(x);
                D_IM(z) = hy >= 0 ? zero : -zero;
        }

#endif /* ! codereview */
        return (z);
}
```

```
**********************************************************
    1482 Thu Jun  6 20:39:51 2019
new/usr/src/lib/libm/common/complex/cprojf.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __cprojf = cprojf

#include "libm.h"
#include "complex_wrapper.h"

/* INDENT OFF */
static const float zero = 0.0F;

/* INDENT ON */

fcomplex
cprojf(fcomplex z)
{
cprojf(fcomplex z) {
        float x, y;
        int ix, iy, hx, hy;

        x = F_RE(z);
        y = F_IM(z);
        hx = THE_WORD(x);
        hy = THE_WORD(y);
        ix = hx & 0x7fffffff;
        iy = hy & 0x7fffffff;

#endif /* ! codereview */
        if (iy == 0x7f800000) {
                F_RE(z) = fabsf(y);
                F_IM(z) = hy >= 0 ? zero : -zero;
        } else if (ix == 0x7f800000) {
                F_RE(z) = fabsf(x);
```



```
                F_IM(z) = hy >= 0 ? zero : -zero;
        }

#endif /* ! codereview */
        return (z);
}
```

```
**********************************************************
    1512 Thu Jun  6 20:39:51 2019
new/usr/src/lib/libm/common/complex/cprojl.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __cprojl = cprojl

#include "libm.h"                       /* fabsl */
#include "complex_wrapper.h"
#include "longdouble.h"

/* INDENT OFF */
static const long double zero = 0.0L;

/* INDENT ON */

ldcomplex
cprojl(ldcomplex z)
{
cprojl(ldcomplex z) {
        long double x, y;
        int hy;

        x = LD_RE(z);
        y = LD_IM(z);
#if defined(__x86)
        hy = ((int *)&y)[2] << 16;
        hy = ((int *) &y)[2] << 16;
#else
        hy = ((int *)&y)[0];
        hy = ((int *) &y)[0];
#endif

#endif /* ! codereview */
        if (isinfl(y)) {
```



```
                LD_RE(z) = fabsl(y);
                LD_IM(z) = hy >= 0 ? zero : -zero;
        } else if (isinfl(x)) {
                LD_RE(z) = fabsl(x);
                LD_IM(z) = hy >= 0 ? zero : -zero;
        }

#endif /* ! codereview */
        return (z);
}
```

```
**********************************************************
    1124 Thu Jun  6 20:39:52 2019
new/usr/src/lib/libm/common/complex/creal.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
   1 /*
   2  * CDDL HEADER START
   3  *
   4  * The contents of this file are subject to the terms of the
   5  * Common Development and Distribution License (the "License").
   6  * You may not use this file except in compliance with the License.
   7  *
   8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
   9  * or http://www.opensolaris.org/os/licensing.
  10  * See the License for the specific language governing permissions
  11  * and limitations under the License.
  12  *
  13  * When distributing Covered Code, include this CDDL HEADER in each
  14  * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  15  * If applicable, add the following below this CDDL HEADER, with the
  16  * fields enclosed by brackets "[]" replaced with your own identifying
  17  * information: Portions Copyright [yyyy] [name of copyright owner]
  18  *
  19  * CDDL HEADER END
  20  */

  22 /*
  23  * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
  24  */

  26 #endif /* ! codereview */
  27 /*
  28  * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
  29  * Use is subject to license terms.
  30  */

  32 #pragma weak __creal = creal

  34 #include "libm.h"
  35 #include "complex_wrapper.h"

  37 double
  38 creal(dcomplex z)
  39 {
  25 creal(dcomplex z) {
  40         return (D_RE(z));
  41 }
______unchanged_portion_omitted_
```

```
**********************************************************
    1126 Thu Jun  6 20:39:52 2019
new/usr/src/lib/libm/common/complex/crealf.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __crealf = crealf

#include "libm.h"
#include "complex_wrapper.h"

float
crealf(fcomplex z)
{
crealf(fcomplex z) {
        return (F_RE(z));
}
______unchanged_portion_omitted_
```

```
**********************************************************
    1134 Thu Jun  6 20:39:53 2019
new/usr/src/lib/libm/common/complex/creall.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __creall = creall

#include "libm.h"
#include "complex_wrapper.h"

long double
creall(ldcomplex z)
{
creall(ldcomplex z) {
        return (LD_RE(z));
}
______unchanged_portion_omitted_
```

```
**********************************************************
    1527 Thu Jun  6 20:39:53 2019
new/usr/src/lib/libm/common/complex/csin.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __csin = csin


/* INDENT OFF */
/*
 * dcomplex csin(dcomplex z);
 *
 * If z = x+iy, then since csin(iz) = i*csinh(z),  we have
 *
 * csin(z)      = csin((-1)*(-z)) = csin(i*i*(-z))
 *              = i*csinh(i*(-z)) = i*csinh(i*(-x-yi))
 *              = i*csinh(y-ix)
 *              = -Im(csinh(y-ix))+i*Re(csinh(y-ix))
 */
/* INDENT ON */

#include "libm.h"
#include "complex_wrapper.h"

dcomplex
csin(dcomplex z)
{
csin(dcomplex z) {
        double x, y;
        dcomplex ans, ct;

        x = D_RE(z);
        y = D_IM(z);
        D_RE(z) = y;
        D_IM(z) = -x;
```



```
        ct = csinh(z);
        D_RE(ans) = -D_IM(ct);
        D_IM(ans) = D_RE(ct);
        return (ans);
}
______unchanged_portion_omitted_
```

```
**********************************************************
    1276 Thu Jun  6 20:39:54 2019
new/usr/src/lib/libm/common/complex/csinf.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __csinf = csinf

#include "libm.h"
#include "complex_wrapper.h"

fcomplex
csinf(fcomplex z)
{
csinf(fcomplex z) {
        float x, y;
        fcomplex ans, ct;

        x = F_RE(z);
        y = F_IM(z);
        F_RE(z) = y;
        F_IM(z) = -x;
        ct = csinhf(z);
        F_RE(ans) = -F_IM(ct);
        F_IM(ans) = F_RE(ct);
        return (ans);
}
______unchanged_portion_omitted_
```

```
**********************************************************
    3896 Thu Jun  6 20:39:54 2019
new/usr/src/lib/libm/common/complex/csinh.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __csinh = csinh


/* INDENT OFF */
/*
 * dcomplex csinh(dcomplex z);
 *
 *             z      -z       x                      -x
 *            e   -  e        e  (cos(y)+i*sin(y)) - e  (cos(-y)+i*sin(-y))
 * sinh z = -------------- =  ---------------------------------------------
 *                2                              2
 *                        x    -x                x    -x
 *            cos(y) ( e  - e  )  + i*sin(y) (e  + e   )
 *        = ---------------------------------------------
 *                                2
 *
 *        =  cos(y) sinh(x)  + i sin(y) cosh(x)
 *
 * Implementation Note
 * -------------------
 *
 *             |x|    -|x|   |x|        -2|x|       -2|x|    -P-4
 * Note that  e   +- e    = e   ( 1 +- e     ). If e      < 2     , where
 *
 * P stands for the number of significant bits of the machine precision,
 *                                      |x|
 * then the result will be rounded to e   . Therefore, we have
 *
 *                 z
 *                e
```



```
 *     sinh z = -----  if |x| >= (P/2 + 2)*ln2
 *                2
 *
 * EXCEPTION (conform to ISO/IEC 9899:1999(E)):
 *      csinh(0,0)=(0,0)
 *      csinh(0,inf)=(+-0,NaN)
 *      csinh(0,NaN)=(+-0,NaN)
 *      csinh(x,inf) = (NaN,NaN) for finite positive x
 *      csinh(x,NaN) = (NaN,NaN) for finite non-zero x
 *      csinh(inf,0) = (inf, 0)
 *      csinh(inf,y) = (inf*cos(y),inf*sin(y)) for positive finite y
 *      csinh(inf,inf) = (+-inf,NaN)
 *      csinh(inf,NaN) = (+-inf,NaN)
 *      csinh(NaN,0) = (NaN,0)
 *      csinh(NaN,y) = (NaN,NaN) for non-zero y
 *      csinh(NaN,NaN) = (NaN,NaN)
 */
/* INDENT ON */

#include "libm.h"       /* cosh/exp/fabs/scalbn/sinh/sincos/__k_cexp */
#include "complex_wrapper.h"

dcomplex
csinh(dcomplex z)
{
csinh(dcomplex z) {
        double t, x, y, S, C;
        int hx, ix, lx, hy, iy, ly, n;
        dcomplex ans;

        x = D_RE(z);
        y = D_IM(z);
        hx = HI_WORD(x);
        lx = LO_WORD(x);
        ix = hx & 0x7fffffff;
        hy = HI_WORD(y);
        ly = LO_WORD(y);
        iy = hy & 0x7fffffff;
        x = fabs(x);
        y = fabs(y);

        (void) sincos(y, &S, &C);

#endif /* ! codereview */
        if (ix >= 0x403c0000) {         /* |x| > 28 = prec/2 (14,28,34,60) */
                if (ix >= 0x40862E42) { /* |x| > 709.78... ~ log(2**1024) */
                        if (ix >= 0x7ff00000) { /* |x| is inf or NaN */
                                if ((iy | ly) == 0) {
                                        D_RE(ans) = x;
                                        D_IM(ans) = y;
                                } else if (iy >= 0x7ff00000) {
                                        D_RE(ans) = x;
                                        D_IM(ans) = x - y;
                                } else {
                                        D_RE(ans) = C * x;
                                        D_IM(ans) = S * x;
                                }
                        } else {
                                /* return exp(x)=t*2**n */
                                t = __k_cexp(x, &n);
                                D_RE(ans) = scalbn(C * t, n - 1);
                                D_IM(ans) = scalbn(S * t, n - 1);
                        }
                } else {
                        t = exp(x) * 0.5;
                        D_RE(ans) = C * t;
```

```
                        D_IM(ans) = S * t;
                }
        } else {
                if ((ix | lx) == 0) {   /* x = 0, return (0,S) */
                        D_RE(ans) = 0.0;
                        D_IM(ans) = S;
                } else {
                        D_RE(ans) = C * sinh(x);
                        D_IM(ans) = S * cosh(x);
                }
        }

#endif /* ! codereview */
        if (hx < 0)
                D_RE(ans) = -D_RE(ans);

#endif /* ! codereview */
        if (hy < 0)
                D_IM(ans) = -D_IM(ans);

#endif /* ! codereview */
        return (ans);
}
```

```
**********************************************************
    2557 Thu Jun  6 20:39:55 2019
new/usr/src/lib/libm/common/complex/csinhf.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

#endif /* ! codereview */
/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2005 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __csinhf = csinhf

#include "libm.h"
#include "complex_wrapper.h"

#if defined(__i386) && !defined(__amd64)
extern int __swapRP(int);
#endif

static const float zero = 0.0F, half = 0.5F;

fcomplex
csinhf(fcomplex z)
{
csinhf(fcomplex z) {
        float x, y, S, C;
        double t;
        int hx, ix, hy, iy, n;
        fcomplex ans;

        x = F_RE(z);
        y = F_IM(z);
        hx = THE_WORD(x);
        ix = hx & 0x7fffffff;
        hy = THE_WORD(y);
        iy = hy & 0x7fffffff;
        x = fabsf(x);
        y = fabsf(y);
```



```
        sincosf(y, &S, &C);

#endif /* ! codereview */
        if (ix >= 0x41600000) {         /* |x| > 14 = prec/2 (14,28,34,60) */
                if (ix >= 0x42B171AA) { /* |x| > 88.722... ~ log(2**128) */
                        if (ix >= 0x7f800000) { /* |x| is inf or NaN */
                                if (iy == 0) {
                                        F_RE(ans) = x;
                                        F_IM(ans) = y;
                                } else if (iy >= 0x7f800000) {
                                        F_RE(ans) = x;
                                        F_IM(ans) = x - y;
                                } else {
                                        F_RE(ans) = C * x;
                                        F_IM(ans) = S * x;
                                }
                        } else {
#if defined(__i386) && !defined(__amd64)
                                int rp = __swapRP(fp_extended);
#endif
                                /* return (C, S) * exp(x) / 2 */
                                t = __k_cexp((double)x, &n);
                                F_RE(ans) = (float)scalbn(C * t, n - 1);
                                F_IM(ans) = (float)scalbn(S * t, n - 1);
#if defined(__i386) && !defined(__amd64)
                                if (rp != fp_extended)
                                        (void) __swapRP(rp);
#endif
                        }
                } else {
                        t = expf(x) * half;
                        F_RE(ans) = C * t;
                        F_IM(ans) = S * t;
                }
        } else {
                if (ix == 0) {          /* x = 0, return (0,S) */
                        F_RE(ans) = zero;
                        F_IM(ans) = S;
                } else {
                        F_RE(ans) = C * sinhf(x);
                        F_IM(ans) = S * coshf(x);
                }
        }

#endif /* ! codereview */
        if (hx < 0)
                F_RE(ans) = -F_RE(ans);

#endif /* ! codereview */
        if (hy < 0)
                F_IM(ans) = -F_IM(ans);

#endif /* ! codereview */
        return (ans);
}
```

```
**********************************************************
    2369 Thu Jun  6 20:39:55 2019
new/usr/src/lib/libm/common/complex/csinhl.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __csinhl = csinhl

#include "libm.h"       /* coshl/expl/fabsl/scalbnl/sincosl/sinhl/__k_cexpl */
#include "complex_wrapper.h"

/* INDENT OFF */
static const long double zero = 0.0L, half = 0.5L;

/* INDENT ON */

ldcomplex
csinhl(ldcomplex z)
{
csinhl(ldcomplex z) {
        long double t, x, y, S, C;
        int hx, ix, hy, iy, n;
        ldcomplex ans;

        x = LD_RE(z);
        y = LD_IM(z);
        hx = HI_XWORD(x);
        ix = hx & 0x7fffffff;
        hy = HI_XWORD(y);
        iy = hy & 0x7fffffff;
        x = fabsl(x);
        y = fabsl(y);

        (void) sincosl(y, &S, &C);

#endif /* ! codereview */
```



```
        if (ix >= 0x4004e000) {         /* |x| > 60 = prec/2 (14,28,34,60) */
                if (ix >= 0x400C62E4) { /* |x| > 11356.52... ~ log(2**16384) */
                        if (ix >= 0x7fff0000) { /* |x| is inf or NaN */
                                if (y == zero) {
                                        LD_RE(ans) = x;
                                        LD_IM(ans) = y;
                                } else if (iy >= 0x7fff0000) {
                                        LD_RE(ans) = x;
                                        LD_IM(ans) = x - y;
                                } else {
                                        LD_RE(ans) = C * x;
                                        LD_IM(ans) = S * x;
                                }
                        } else {
                                /* return exp(x)=t*2**n */
                                t = __k_cexpl(x, &n);
                                LD_RE(ans) = scalbnl(C * t, n - 1);
                                LD_IM(ans) = scalbnl(S * t, n - 1);
                        }
                } else {
                        t = expl(x) * half;
                        LD_RE(ans) = C * t;
                        LD_IM(ans) = S * t;
                }
        } else {
                if (x == zero) {        /* x = 0, return (0,S) */
                        LD_RE(ans) = zero;
                        LD_IM(ans) = S;
                } else {
                        LD_RE(ans) = C * sinhl(x);
                        LD_IM(ans) = S * coshl(x);
                }
        }

#endif /* ! codereview */
        if (hx < 0)
                LD_RE(ans) = -LD_RE(ans);

#endif /* ! codereview */
        if (hy < 0)
                LD_IM(ans) = -LD_IM(ans);

#endif /* ! codereview */
        return (ans);
}
```

```
**********************************************************
    1293 Thu Jun  6 20:39:56 2019
new/usr/src/lib/libm/common/complex/csinl.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __csinl = csinl

#include "libm.h"
#include "complex_wrapper.h"

ldcomplex
csinl(ldcomplex z)
{
csinl(ldcomplex z) {
        long double x, y;
        ldcomplex ans, ct;

        x = LD_RE(z);
        y = LD_IM(z);
        LD_RE(z) = y;
        LD_IM(z) = -x;
        ct = csinhl(z);
        LD_RE(ans) = -LD_IM(ct);
        LD_IM(ans) = LD_RE(ct);
        return (ans);
}
______unchanged_portion_omitted_
```

```
**********************************************************
    5634 Thu Jun  6 20:39:56 2019
new/usr/src/lib/libm/common/complex/csqrt.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __csqrt = csqrt


/* INDENT OFF */
/*
 * dcomplex csqrt(dcomplex z);
 *
 *                                          2    2    2
 * Let w=r+i*s = sqrt(x+iy). Then (r + i s)  = r  - s  + i 2sr = x + i y.
 *
 * Hence x = r*r-s*s, y = 2sr.
 *
 * Note that x*x+y*y = (s*s+r*r)**2. Thus, we have
 *                                  _______
 *            2    2     / 2    2
 *      (1) r  + s  = \/ x  + y  ,
 *
 *            2    2
 *       (2) r  - s  = x
 *
 *      (3) 2sr = y.
 *
 * Perform (1)-(2) and (1)+(2), we obtain
 *
 *             2
 *      (4) 2 r   = hypot(x,y)+x,
 *
 *             2
 *       (5) 2*s   = hypot(x,y)-x
 *                        _______
```



```
 *                       / 2    2
 * where hypot(x,y) = \/ x  + y  .
 *
 * In order to avoid numerical cancellation, we use formula (4) for
 * positive x, and (5) for negative x. The other component is then
 * computed by formula (3).
 *
 *
 * ALGORITHM
 * -----------------
 *
 * (assume x and y are of medium size, i.e., no over/underflow in squaring)
 *
 * If x >=0 then
 *                          _______
 *                       /  2    2
 *             2     \/  x  + y    +  x                y
 *            r =   ---------------------,      s = -------;    (6)
 *                                2                     2 r
 *
 * (note that we choose sign(s) = sign(y) to force r >=0).
 * Otherwise,
 *                          _______
 *                       /  2    2
 *             2     \/  x  + y    -  x                y
 *            s =   ---------------------,      r = -------;    (7)
 *                                2                     2 s
 *
 * EXCEPTION:
 *
 * One may use the polar coordinate of a complex number to justify the
 * following exception cases:
 *
 * EXCEPTION CASES (conform to ISO/IEC 9899:1999(E)):
 *    csqrt(+-0+ i 0   ) =  0    + i 0
 *    csqrt( x + i inf ) =  inf  + i inf for all x (including NaN)
 *    csqrt( x + i NaN ) =  NaN  + i NaN with invalid for finite x
 *    csqrt(-inf+ iy   ) =  0    + i inf for finite positive-signed y
 *    csqrt(+inf+ iy   ) =  inf  + i 0   for finite positive-signed y
 *    csqrt(-inf+ i NaN) =  NaN  +-i inf
 *    csqrt(+inf+ i NaN) =  inf  + i NaN
 *    csqrt(NaN + i y  ) =  NaN  + i NaN for finite y
 *    csqrt(NaN + i NaN) =  NaN  + i NaN
 */
/* INDENT ON */

#include "libm.h"                       /* fabs/sqrt */
#include "complex_wrapper.h"

static const double two300 = 2.03703597633448608627e+90,
/* INDENT OFF */
static const double
        two300 = 2.03703597633448608627e+90,
        twom300 = 4.90909346529772655310e-91,
        two599 = 2.07475778444049647926e+180,
        twom601 = 1.20495993255144205887e-181,
        two = 2.0,
        zero = 0.0,
        half = 0.5;

/* INDENT ON */

dcomplex
csqrt(dcomplex z)
{
csqrt(dcomplex z) {
```

```
121         dcomplex ans;
122         double x, y, t, ax, ay;
123         int n, ix, iy, hx, hy, lx, ly;

125         x = D_RE(z);
126         y = D_IM(z);
127         hx = HI_WORD(x);
128         lx = LO_WORD(x);
129         hy = HI_WORD(y);
130         ly = LO_WORD(y);
131         ix = hx & 0x7fffffff;
132         iy = hy & 0x7fffffff;
133         ay = fabs(y);
134         ax = fabs(x);

136 #endif /* ! codereview */
137         if (ix >= 0x7ff00000 || iy >= 0x7ff00000) {
138                 /* x or y is Inf or NaN */
139                 if (ISINF(iy, ly)) {
128                 if (ISINF(iy, ly))
140                         D_IM(ans) = D_RE(ans) = ay;
141                 } else if (ISINF(ix, lx)) {
130                 else if (ISINF(ix, lx)) {
142                         if (hx > 0) {
143                                 D_RE(ans) = ax;
144                                 D_IM(ans) = ay * zero;
145                         } else {
146                                 D_RE(ans) = ay * zero;
147                                 D_IM(ans) = ax;
148                         }
149                 } else {
138                 } else
150                         D_IM(ans) = D_RE(ans) = ax + ay;
151                 }
152 #endif /* ! codereview */
153         } else if ((iy | ly) == 0) {    /* y = 0 */
154                 if (hx >= 0) {
155                         D_RE(ans) = sqrt(ax);
156                         D_IM(ans) = zero;
157                 } else {
158                         D_IM(ans) = sqrt(ax);
159                         D_RE(ans) = zero;
160                 }
161         } else if (ix >= iy) {
162                 n = (ix - iy) >> 20;

164 #endif /* ! codereview */
165                 if (n >= 30) {                  /* x >> y or y=0 */
166                         t = sqrt(ax);
167                 } else if (ix >= 0x5f300000) {  /* x > 2**500 */
168                         ax *= twom601;
169                         y *= twom601;
170                         t = two300 * sqrt(ax + sqrt(ax * ax + y * y));
171                 } else if (iy < 0x20b00000) {   /* y < 2**-500 */
172                         ax *= two599;
173                         y *= two599;
174                         t = twom300 * sqrt(ax + sqrt(ax * ax + y * y));
175                 } else {
140                 } else
176                         t = sqrt(half * (ax + sqrt(ax * ax + ay * ay)));
177                 }

179 #endif /* ! codereview */
180                 if (hx >= 0) {
181                         D_RE(ans) = t;
182                         D_IM(ans) = ay / (t + t);
```



```
183                 } else {
184                         D_IM(ans) = t;
185                         D_RE(ans) = ay / (t + t);
186                 }
187         } else {
188                 n = (iy - ix) >> 20;

190 #endif /* ! codereview */
191                 if (n >= 30) {          /* y >> x */
192                         if (n >= 60)
193                                 t = sqrt(half * ay);
194                         else if (iy >= 0x7fe00000)
195                                 t = sqrt(half * ay + half * ax);
196                         else if (ix <= 0x00100000)
197                                 t = half * sqrt(two * (ay + ax));
198                         else
199                                 t = sqrt(half * (ay + ax));
200                 } else if (iy >= 0x5f300000) {  /* y > 2**500 */
201                         ax *= twom601;
202                         y *= twom601;
203                         t = two300 * sqrt(ax + sqrt(ax * ax + y * y));
204                 } else if (ix < 0x20b00000) {   /* x < 2**-500 */
205                         ax *= two599;
206                         y *= two599;
207                         t = twom300 * sqrt(ax + sqrt(ax * ax + y * y));
208                 } else {
142                 } else
209                         t = sqrt(half * (ax + sqrt(ax * ax + ay * ay)));
210                 }

212 #endif /* ! codereview */
213                 if (hx >= 0) {
214                         D_RE(ans) = t;
215                         D_IM(ans) = ay / (t + t);
216                 } else {
217                         D_IM(ans) = t;
218                         D_RE(ans) = ay / (t + t);
219                 }
220         }

222 #endif /* ! codereview */
223         if (hy < 0)
224                 D_IM(ans) = -D_IM(ans);

226 #endif /* ! codereview */
227         return (ans);
228 }
```

```
**********************************************************
    2201 Thu Jun  6 20:39:57 2019
new/usr/src/lib/libm/common/complex/csqrtf.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __csqrtf = csqrtf

#include "libm.h"                       /* sqrt/fabsf/sqrtf */
#include "complex_wrapper.h"

/* INDENT OFF */
static const float zero = 0.0F;

/* INDENT ON */

fcomplex
csqrtf(fcomplex z)
{
csqrtf(fcomplex z) {
        fcomplex ans;
        double dt, dx, dy;
        float x, y, t, ax, ay, w;
        int ix, iy, hx, hy;

        x = F_RE(z);
        y = F_IM(z);
        hx = THE_WORD(x);
        hy = THE_WORD(y);
        ix = hx & 0x7fffffff;
        iy = hy & 0x7fffffff;
        ay = fabsf(y);
        ax = fabsf(x);

#endif /* ! codereview */
        if (ix >= 0x7f800000 || iy >= 0x7f800000) {
```



```
                /* x or y is Inf or NaN */
                if (iy == 0x7f800000) {
                if (iy == 0x7f800000)
                        F_IM(ans) = F_RE(ans) = ay;
                } else if (ix == 0x7f800000) {
                else if (ix == 0x7f800000) {
                        if (hx > 0) {
                                F_RE(ans) = ax;
                                F_IM(ans) = ay * zero;
                        } else {
                                F_RE(ans) = ay * zero;
                                F_IM(ans) = ax;
                        }
                } else {
                } else
                        F_IM(ans) = F_RE(ans) = ax + ay;
                }
#endif /* ! codereview */
        } else if (iy == 0) {
                if (hx >= 0) {
                        F_RE(ans) = sqrtf(ax);
                        F_IM(ans) = zero;
                } else {
                        F_IM(ans) = sqrtf(ax);
                        F_RE(ans) = zero;
                }
        } else {
                dx = (double)ax;
                dy = (double)ay;
                dx = (double) ax;
                dy = (double) ay;
                dt = sqrt(0.5 * (sqrt(dx * dx + dy * dy) + dx));
                t = (float)dt;
                w = (float)(dy / (dt + dt));

                t = (float) dt;
                w = (float) (dy / (dt + dt));
                if (hx >= 0) {
                        F_RE(ans) = t;
                        F_IM(ans) = w;
                } else {
                        F_IM(ans) = t;
                        F_RE(ans) = w;
                }
        }

#endif /* ! codereview */
        if (hy < 0)
                F_IM(ans) = -F_IM(ans);

#endif /* ! codereview */
        return (ans);
}
```

```
**********************************************************
    3768 Thu Jun  6 20:39:57 2019
new/usr/src/lib/libm/common/complex/csqrtl.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __csqrtl = csqrtl

#include "libm.h"                       /* fabsl/isinfl/sqrtl */
#include "complex_wrapper.h"
#include "longdouble.h"

/* INDENT OFF */
static const long double
        twom9001 = 2.685400271600303495742176510061569304365 6e-2710L,
        twom4500 = 2.317498768759242942326324286238154414925 2e-1355L,
        two8999 = 9.309599118012234350258234737216329031093 4e+2708L,
        two4500 = 4.314996898727097428377780354557172225080 6e+1354L,
        zero = 0.0L,
        half = 0.5L,
        two = 2.0L;

/* INDENT ON */

ldcomplex
csqrtl(ldcomplex z)
{
csqrtl(ldcomplex z) {
        ldcomplex ans;
        long double x, y, t, ax, ay;
        int n, ix, iy, hx, hy;

        x = LD_RE(z);
        y = LD_IM(z);
        hx = HI_XWORD(x);
        hy = HI_XWORD(y);
```



```
        ix = hx & 0x7fffffff;
        iy = hy & 0x7fffffff;
        ay = fabsl(y);
        ax = fabsl(x);

#endif /* ! codereview */
        if (ix >= 0x7fff0000 || iy >= 0x7fff0000) {
                /* x or y is Inf or NaN */
                if (isinfl(y)) {
                if (isinfl(y))
                        LD_IM(ans) = LD_RE(ans) = ay;
                } else if (isinfl(x)) {
                else if (isinfl(x)) {
                        if (hx > 0) {
                                LD_RE(ans) = ax;
                                LD_IM(ans) = ay * zero;
                        } else {
                                LD_RE(ans) = ay * zero;
                                LD_IM(ans) = ax;
                        }
                } else {
                } else
                        LD_IM(ans) = LD_RE(ans) = ax + ay;
                }
#endif /* ! codereview */
        } else if (y == zero) {
                if (hx >= 0) {
                        LD_RE(ans) = sqrtl(ax);
                        LD_IM(ans) = zero;
                } else {
                        LD_IM(ans) = sqrtl(ax);
                        LD_RE(ans) = zero;
                }
        } else if (ix >= iy) {
                n = (ix - iy) >> 16;
#if defined(__x86)                      /* 64 significant bits */
                if (n >= 35)
#else /* 113 significant bits  */
                if (n >= 60)
#endif
                {
#endif /* ! codereview */
                        t = sqrtl(ax);
                } else if (ix >= 0x5f3f0000) {  /* x > 2**8000 */
                else if (ix >= 0x5f3f0000) {    /* x > 2**8000 */
                        ax *= twom9001;
                        y *= twom9001;
                        t = two4500 * sqrtl(ax + sqrtl(ax * ax + y * y));
                } else if (iy <= 0x20bf0000) {  /* y < 2**-8000 */
                        ax *= two8999;
                        y *= two8999;
                        t = twom4500 * sqrtl(ax + sqrtl(ax * ax + y * y));
                } else {
                } else
                        t = sqrtl(half * (ax + sqrtl(ax * ax + y * y)));
                }
#endif /* ! codereview */

                if (hx >= 0) {
                        LD_RE(ans) = t;
                        LD_IM(ans) = ay / (t + t);
                } else {
                        LD_IM(ans) = t;
                        LD_RE(ans) = ay / (t + t);
                }
        } else {
```

```
                n = (iy - ix) >> 16;
#if defined(__x86)                      /* 64 significant bits */
                if (n >= 35) {          /* } */
#else /* 113 significant bits  */
                if (n >= 60) {
#endif

#endif /* ! codereview */
                        if (n >= 120)
                                t = sqrtl(half * ay);
                        else if (iy >= 0x7ffe0000)
                                t = sqrtl(half * ay + half * ax);
                        else if (ix <= 0x00010000)
                                t = half * (sqrtl(two * (ax + ay)));
                        else
                                t = sqrtl(half * (ax + ay));
                } else if (iy >= 0x5f3f0000) {  /* y > 2**8000 */
                        ax *= twom9001;
                        y *= twom9001;
                        t = two4500 * sqrtl(ax + sqrtl(ax * ax + y * y));
                } else if (ix <= 0x20bf0000) {
                        ax *= two8999;
                        y *= two8999;
                        t = twom4500 * sqrtl(ax + sqrtl(ax * ax + y * y));
                } else {
                } else
                        t = sqrtl(half * (ax + sqrtl(ax * ax + y * y)));
                }
#endif /* ! codereview */

                if (hx >= 0) {
                        LD_RE(ans) = t;
                        LD_IM(ans) = ay / (t + t);
                } else {
                        LD_IM(ans) = t;
                        LD_RE(ans) = ay / (t + t);
                }
        }

#endif /* ! codereview */
        if (hy < 0)
                LD_IM(ans) = -LD_IM(ans);

#endif /* ! codereview */
        return (ans);
}
```

```
**********************************************************
    1527 Thu Jun  6 20:39:58 2019
new/usr/src/lib/libm/common/complex/ctan.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __ctan = ctan


/* INDENT OFF */
/*
 * dcomplex ctan(dcomplex z);
 *
 * If z = x+iy, then since ctan(iz) = i*ctanh(z),  we have
 *
 * ctan(z)      = ctan((-1)*(-z)) = ctan(i*i*(-z))
 *              = i*ctanh(i*(-z)) = i*ctanh(i*(-x-yi))
 *              = i*ctanh(y-ix)
 *              = -Im(ctanh(y-ix))+i*Re(ctanh(y-ix))
 */
/* INDENT ON */

#include "libm.h"
#include "complex_wrapper.h"

dcomplex
ctan(dcomplex z)
{
ctan(dcomplex z) {
        double x, y;
        dcomplex ans, ct;

        x = D_RE(z);
        y = D_IM(z);
        D_RE(z) = y;
        D_IM(z) = -x;
```



```
        ct = ctanh(z);
        D_RE(ans) = -D_IM(ct);
        D_IM(ans) = D_RE(ct);
        return (ans);
}
```

*_____unchanged_portion_omitted_*

```
**********************************************************
    1276 Thu Jun  6 20:39:58 2019
new/usr/src/lib/libm/common/complex/ctanf.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __ctanf = ctanf

#include "libm.h"
#include "complex_wrapper.h"

fcomplex
ctanf(fcomplex z)
{
ctanf(fcomplex z) {
        float x, y;
        fcomplex ans, ct;

        x = F_RE(z);
        y = F_IM(z);
        F_RE(z) = y;
        F_IM(z) = -x;
        ct = ctanhf(z);
        F_RE(ans) = -F_IM(ct);
        F_IM(ans) = F_RE(ct);
        return (ans);
}
______unchanged_portion_omitted_
```

```
**********************************************************
    5405 Thu Jun  6 20:39:59 2019
new/usr/src/lib/libm/common/complex/ctanh.c
11210 libm should be cstyle(10NBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __ctanh = ctanh


/* INDENT OFF */
/*
 * dcomplex ctanh(dcomplex z);
 *
 *            tanh x  + i tan y      sinh 2x  +  i sin 2y
 * ctanh z = --------------------- = --------------------
 *           1 + i tanh(x)tan(y)       cosh 2x + cos 2y
 *
 * For |x| >= prec/2 (14,28,34,60 for single, double, double extended, quad),
 * we use
 *
 *                        1   2x                                2 sin 2y
 *    cosh 2x = sinh 2x = --- e    and hence  ctanh z = 1 + i -----------;
 *                        2                                        2x
 *                                                               e
 *
 * otherwise, to avoid cancellation, for |x| < prec/2,
 *                                 2x     2
 *                             (e   - 1)        2       2
 *    cosh 2x + cos 2y = 1 + ------------ + cos y - sin y
 *                                 2x
 *                               2 e
 *
 *                        1    2x     2  -2x         2
 *                     = --- (e   - 1)  e     + 2 cos y
 *                        2
 * and
 *
 *                   [            2x      ]
 *                 1 [  2x       e   - 1  ]
 *    sinh 2x = --- [ e  - 1 + --------- ]
 *                 2 [               2x   ]
 *                   [              e     ]
 *                                                 2x
 * Implementation notes:  let t = expm1(2x) = e   - 1, then
 *
 *                      1    [  t*t         2  ]              1   [      t  ]
 * cosh 2x + cos 2y = --- * [ ----- + 4 cos y ];  sinh 2x = --- * [ t + --- ]
 *                      2    [  t+1            ]              2    [     t+1 ]
 *
 * Hence,
 *
 *
 *                          t*t+2t                  [4(t+1)(cos y)]*(sin y)
 *    ctanh z = -------------------------- + i -------------------------
 *                t*t+[4(t+1)(cos y)](cos y)     t*t+[4(t+1)(cos y)](cos y)
 *
 * EXCEPTION (conform to ISO/IEC 9899:1999(E)):
 *      ctanh(0,0)=(0,0)
 *      ctanh(x,inf) = (NaN,NaN) for finite x
 *      ctanh(x,NaN) = (NaN,NaN) for finite x
 *      ctanh(inf,y) = 1+ i*0*sin(2y) for positive-signed finite y
 *      ctanh(inf,inf) = (1, +-0)
 *      ctanh(inf,NaN) = (1, +-0)
 *      ctanh(NaN,0) = (NaN,0)
 *      ctanh(NaN,y) = (NaN,NaN) for non-zero y
 *      ctanh(NaN,NaN) = (NaN,NaN)
 */
/* INDENT ON */

#include "libm.h"                       /* exp/expm1/fabs/sin/tanh/sincos */
#include "complex_wrapper.h"

static const double four = 4.0, two = 2.0, one = 1.0, zero = 0.0;

dcomplex
ctanh(dcomplex z)
{
ctanh(dcomplex z) {
        double t, r, v, u, x, y, S, C;
        int hx, ix, lx, hy, iy, ly;
        dcomplex ans;

        x = D_RE(z);
        y = D_IM(z);
        hx = HI_WORD(x);
        lx = LO_WORD(x);
        ix = hx & 0x7fffffff;
        hy = HI_WORD(y);
        ly = LO_WORD(y);
        iy = hy & 0x7fffffff;
        x = fabs(x);
        y = fabs(y);

        if ((iy | ly) == 0) {   /* ctanh(x,0) = (x,0) for x = 0 or NaN */
                D_RE(ans) = tanh(x);
                D_IM(ans) = zero;
        } else if (iy >= 0x7ff00000) { /* y is inf or NaN */
                if (ix < 0x7ff00000) { /* catanh(finite x,inf/nan) is nan */
                if (ix < 0x7ff00000)    /* catanh(finite x,inf/nan) is nan */
                        D_RE(ans) = D_IM(ans) = y - y;
                } else if (((ix - 0x7ff00000) | lx) == 0) {     /* x is inf */
                else if (((ix - 0x7ff00000) | lx) == 0) {       /* x is inf */
```

```
                        D_RE(ans) = one;
                        D_IM(ans) = zero;
                } else {
                        D_RE(ans) = x + y;
                        D_IM(ans) = y - y;
                }
        } else if (ix >= 0x403c0000) {
                /*
                 * |x| > 28 = prec/2 (14,28,34,60)
                 * ctanh z ~ 1 + i (sin2y)/(exp(2x))
                 */
                D_RE(ans) = one;

                if (iy < 0x7fe00000) {  /* t = sin(2y) */
                if (iy < 0x7fe00000)    /* t = sin(2y) */
                        S = sin(y + y);
                } else {
                else {
                        (void) sincos(y, &S, &C);
                        S = (S + S) * C;
                }

#endif /* ! codereview */
                if (ix >= 0x7fe00000) {         /* |x| > max/2 */
                        if (ix >= 0x7ff00000) { /* |x| is inf or NaN */
                                if (((ix - 0x7ff00000) | lx) != 0)
                                        D_RE(ans) = D_IM(ans) = x + y;
                                /* x is NaN */
                                else
                                        D_IM(ans) = zero * S;   /* x is inf */
                        } else {
                        } else
                                D_IM(ans) = S * exp(-x);        /* underflow */
                        }
                } else {
                } else
                        D_IM(ans) = (S + S) * exp(-(x + x));
                }

#endif /* ! codereview */
                /* 2 sin 2y / exp(2x) */
        } else {
                /* BEGIN CSTYLED */
                /* INDENT OFF */
                /*
                 *                        t*t+2t
                 *    ctanh z = -------------------------- +
                 *               t*t+[4(t+1)(cos y)](cos y)
                 *
                 *                  [4(t+1)(cos y)]*(sin y)
                 *              i -------------------------
                 *                t*t+[4(t+1)(cos y)](cos y)
                 */
                /* END CSTYLED */
                /* INDENT ON */
                (void) sincos(y, &S, &C);
                t = expm1(x + x);
                r = (four * C) * (t + one);
                u = t * t;
                v = one / (u + r * C);
                D_RE(ans) = (u + two * t) * v;
                D_IM(ans) = (r * S) * v;
        }

#endif /* ! codereview */
        if (hx < 0)
```



```
                D_RE(ans) = -D_RE(ans);

#endif /* ! codereview */
        if (hy < 0)
                D_IM(ans) = -D_IM(ans);

#endif /* ! codereview */
        return (ans);
}
```

```
**********************************************************
    3088 Thu Jun  6 20:39:59 2019
new/usr/src/lib/libm/common/complex/ctanhf.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __ctanhf = ctanhf

#include "libm.h"       /* expf/expm1f/fabsf/sincosf/sinf/tanhf */
#include "complex_wrapper.h"

/* INDENT OFF */
static const float four = 4.0F, two = 2.0F, one = 1.0F, zero = 0.0F;

/* INDENT ON */

fcomplex
ctanhf(fcomplex z)
{
ctanhf(fcomplex z) {
        float r, u, v, t, x, y, S, C;
        int hx, ix, hy, iy;
        fcomplex ans;

        x = F_RE(z);
        y = F_IM(z);
        hx = THE_WORD(x);
        ix = hx & 0x7fffffff;
        hy = THE_WORD(y);
        iy = hy & 0x7fffffff;
        x = fabsf(x);
        y = fabsf(y);

        if (iy == 0) {          /* ctanh(x,0) = (x,0) for x = 0 or NaN */
                F_RE(ans) = tanhf(x);
                F_IM(ans) = zero;
```



```
        } else if (iy >= 0x7f800000) { /* y is inf or NaN */
                if (ix < 0x7f800000) { /* catanh(finite x,inf/nan) is nan */
                if (ix < 0x7f800000)    /* catanh(finite x,inf/nan) is nan */
                        F_RE(ans) = F_IM(ans) = y - y;
                } else if (ix == 0x7f800000) {  /* x is inf */
                else if (ix == 0x7f800000) {    /* x is inf */
                        F_RE(ans) = one;
                        F_IM(ans) = zero;
                } else {
                        F_RE(ans) = x + y;
                        F_IM(ans) = y - y;
                }
        } else if (ix >= 0x41600000) {
                /*
                 * |x| > 14 = prec/2 (14,28,34,60)
                 * ctanh z ~ 1 + i (sin2y)/(exp(2x))
                 */
                F_RE(ans) = one;

                if (iy < 0x7f000000) {  /* t = sin(2y) */
                if (iy < 0x7f000000)    /* t = sin(2y) */
                        S = sinf(y + y);
                } else {
                else {
                        (void) sincosf(y, &S, &C);
                        S = (S + S) * C;
                }

#endif /* ! codereview */
                if (ix >= 0x7f000000) {         /* |x| > max/2 */
                        if (ix >= 0x7f800000) { /* |x| is inf or NaN */
                                if (ix > 0x7f800000)            /* x is NaN */
                                        F_RE(ans) = F_IM(ans) = x + y;
                                else
                                        F_IM(ans) = zero * S;   /* x is inf */
                        } else {
                        } else
                                F_IM(ans) = S * expf(-x);       /* underflow */
                        }
                } else {
                } else
                        F_IM(ans) = (S + S) * expf(-(x + x));
                }

#endif /* ! codereview */
                /* 2 sin 2y / exp(2x) */
        } else {
                /* BEGIN CSTYLED */
                /* INDENT OFF */
                /*
                 *                        t*t+2t
                 *    ctanh z = ---------------------------
                 *                t*t+[4(t+1)(cos y)](cos y)
                 *
                 *                    [4(t+1)(cos y)]*(sin y)
                 *              i --------------------------
                 *                t*t+[4(t+1)(cos y)](cos y)
                 */
                /* END CSTYLED */
                /* INDENT ON */
                (void) sincosf(y, &S, &C);
                t = expm1f(x + x);
                r = (four * C) * (t + one);
                u = t * t;
                v = one / (u + r * C);
                F_RE(ans) = (u + two * t) * v;
```

```
                F_IM(ans) = (r * S) * v;
        }

#endif /* ! codereview */
        if (hx < 0)
                F_RE(ans) = -F_RE(ans);

#endif /* ! codereview */
        if (hy < 0)
                F_IM(ans) = -F_IM(ans);

#endif /* ! codereview */
        return (ans);
}
```

```
**********************************************************
    3144 Thu Jun  6 20:40:00 2019
new/usr/src/lib/libm/common/complex/ctanhl.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __ctanhl = ctanhl

#include "libm.h"       /* expl/expm1l/fabsl/isinfl/isnanl/sincosl/sinl/tanhl */
#include "complex_wrapper.h"
#include "longdouble.h"

/* INDENT OFF */
static const long double four = 4.0L, two = 2.0L, one = 1.0L, zero = 0.0L;

/* INDENT ON */

ldcomplex
ctanhl(ldcomplex z)
{
ctanhl(ldcomplex z) {
        long double r, u, v, t, x, y, S, C;
        int hx, ix, hy, iy;
        ldcomplex ans;

        x = LD_RE(z);
        y = LD_IM(z);
        hx = HI_XWORD(x);
        ix = hx & 0x7fffffff;
        hy = HI_XWORD(y);
        iy = hy & 0x7fffffff;
        x = fabsl(x);
        y = fabsl(y);

        if (y == zero) {        /* ctanh(x,0) = (x,0) for x = 0 or NaN */
                LD_RE(ans) = tanhl(x);
```



```
                LD_IM(ans) = zero;
        } else if (iy >= 0x7fff0000) {  /* y is inf or NaN */
                if (ix < 0x7fff0000) {  /* catanh(finite x,inf/nan) is nan */
                if (ix < 0x7fff0000)    /* catanh(finite x,inf/nan) is nan */
                        LD_RE(ans) = LD_IM(ans) = y - y;
                } else if (isinfl(x)) { /* x is inf */
                else if (isinfl(x)) {   /* x is inf */
                        LD_RE(ans) = one;
                        LD_IM(ans) = zero;
                } else {
                        LD_RE(ans) = x + y;
                        LD_IM(ans) = y - y;
                }
        } else if (ix >= 0x4004e000) {

                /* INDENT OFF */
                /*
                 * |x| > 60 = prec/2 (14,28,34,60)
                 * ctanh z ~ 1 + i (sin2y)/(exp(2x))
                 */
                /* INDENT ON */
                LD_RE(ans) = one;

                if (iy < 0x7ffe0000) {  /* t = sin(2y) */
                if (iy < 0x7ffe0000)    /* t = sin(2y) */
                        S = sinl(y + y);
                } else {
                else {
                        (void) sincosl(y, &S, &C);
                        S = (S + S) * C;
                }

#endif /* ! codereview */
                if (ix >= 0x7ffe0000) {         /* |x| > max/2 */
                        if (ix >= 0x7fff0000) { /* |x| is inf or NaN */
                                if (isnanl(x))  /* x is NaN */
                                        LD_RE(ans) = LD_IM(ans) = x + y;
                                else
                                        LD_IM(ans) = zero * S;  /* x is inf */
                        } else {
                        } else
                                LD_IM(ans) = S * expl(-x);      /* underflow */
                        }
                } else {
                } else
                        LD_IM(ans) = (S + S) * expl(-(x + x));
                }

#endif /* ! codereview */
                /* 2 sin 2y / exp(2x) */
        } else {
                /* BEGIN CSTYLED */
                /* INDENT OFF */
                /*
                 *                        t*t+2t
                 *    ctanh z = ---------------------------
                 *              t*t+[4(t+1)(cos y)](cos y)
                 *
                 *                 [4(t+1)(cos y)]*(sin y)
                 *              i ---------------------------
                 *                t*t+[4(t+1)(cos y)](cos y)
                 */
                /* END CSTYLED */
                /* INDENT ON */
                sincosl(y, &S, &C);
                t = expm1l(x + x);
```

```
                r = (four * C) * (t + one);
                u = t * t;
                v = one / (u + r * C);
                LD_RE(ans) = (u + two * t) * v;
                LD_IM(ans) = (r * S) * v;
        }

#endif /* ! codereview */
        if (hx < 0)
                LD_RE(ans) = -LD_RE(ans);

#endif /* ! codereview */
        if (hy < 0)
                LD_IM(ans) = -LD_IM(ans);

#endif /* ! codereview */
        return (ans);
}
```

```
**********************************************************
    1293 Thu Jun  6 20:40:00 2019
new/usr/src/lib/libm/common/complex/ctanl.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __ctanl = ctanl

#include "libm.h"
#include "complex_wrapper.h"

ldcomplex
ctanl(ldcomplex z)
{
ctanl(ldcomplex z) {
        long double x, y;
        ldcomplex ans, ct;

        x = LD_RE(z);
        y = LD_IM(z);
        LD_RE(z) = y;
        LD_IM(z) = -x;
        ct = ctanhl(z);
        LD_RE(ans) = -LD_IM(ct);
        LD_IM(ans) = LD_RE(ct);
        return (ans);
}
______unchanged_portion_omitted_
```

```
**********************************************************
   18383 Thu Jun  6 20:40:01 2019
new/usr/src/lib/libm/common/complex/k_atan2.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

#endif /* ! codereview */
/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2005 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#include "libm.h"                       /* __k_atan2 */
#include "complex_wrapper.h"

/*
 * double __k_atan2(double y, double x, double *e)
 *
 * Compute atan2 with error terms.
 *
 * Important formula:
 *                  3       5
 *                 x       x
 * atan(x) = x - ----- + ----- - ...    (for x <= 1)
 *                 3       5
 *
 *           pi     1     1
 *         = --- - --- + --- - ...      (for x > 1)
 *                   3
 *            2     x    3x
 *
 * Arg(x + y i) = sign(y) * atan2(|y|, x)
 *              = sign(y) * atan(|y|/x)          (for x > 0)
 *                sign(y) * (PI - atan(|y|/|x|))  (for x < 0)
 * Thus if x >> y (IEEE double: EXP(x) - EXP(y) >= 60):
 *      1. (x > 0): atan2(y,x) ~ y/x
 *      2. (x < 0): atan2(y,x) ~ sign(y) (PI - |y/x|))
 * Otherwise if x << y:
 *      atan2(y,x) ~ sign(y)*PI/2 - x/y
 *
 * __k_atan2 call static functions mx_poly, mx_atan
```



```
 */

/*
 * (void) mx_poly (double *z, double *a, double *e, int n)
 * return
 *      e = a  + z*(a  + z*(a  + ... z*(a  + e)...))
 *           0       2       4           2n
 * Note:
 * 1.   e and coefficient ai are represented by two double numbers.
 *      For e, the first one contain the leading 24 bits rounded, and the
 *      second one contain the remaining 53 bits (total 77 bits accuracy).
 *      For ai, the first one contian the leading 53 bits rounded, and the
 *      second is the remaining 53 bits (total 106 bits accuracy).
 * 2.   z is an array of three doubles.
 *      z[0] :  the rounded value of Z (the intended value of z)
 *      z[1] :  the leading 24 bits of Z rounded
 *      z[2] :  the remaining 53 bits of Z
 *              Note that z[0] = z[1]+z[2] rounded.
 *
 */

static void
mx_poly(const double *z, const double *a, double *e, int n)
{
mx_poly(const double *z, const double *a, double *e, int n) {
        double r, s, t, p_h, p_l, z_h, z_l, p;
        int i;

        n = n + n;
        p = e[0] + a[n];
        p_l = a[n + 1];
        p_h = (double)((float)p);
        p_h = (double) ((float) p);
        p = a[n - 2] + z[0] * p;
        z_h = z[1];
        z_l = z[2];
        z_h = z[1]; z_l = z[2];
        p_l += e[0] - (p_h - a[n]);

        for (i = n - 2; i >= 2; i -= 2) {
                /* compute p = ai + z * p */
                t = z_h * p_h;
                s = z[0] * p_l + p_h * z_l;
                p_h = (double)((float)p);
                p_h = (double) ((float) p);
                s += a[i + 1];
                r = t - (p_h - a[i]);
                p = a[i - 2] + z[0] * p;
                p_l = r + s;
        }

        e[0] = (double)((float)p);
        e[0] = (double)((float) p);
        t = z_h * p_h;
        s = z[0] * p_l + p_h * z_l;
        r = t - (e[0] - a[0]);
        e[1] = r + s;
}

/*
 * Table of constants for atan from 0.125 to 8
 *      0.125 -- 0x3fc00000  --- (increment at bit 16)
 *               0x3fc10000
 *               0x3fc20000
 *      ...      ...
 *               0x401f0000
```

```
 *      8.000 -- 0x40200000      (total: 97)
 * By K.C. Ng, March 9, 1989
 */

static const double TBL_atan_hi[] = {
        1.243549945467614382e-01, 1.320397616146387620e-01,
        1.397088742891636204e-01, 1.473614810886516302e-01,
        1.549967419239409727e-01, 1.626138285979485676e-01,
        1.702119252854744080e-01, 1.777902289926760471e-01,
        1.853479499956947607e-01, 1.928843122579746439e-01,
        2.003985538258785115e-01, 2.078899272022629863e-01,
        2.153576996977380476e-01, 2.228011537593945213e-01,
        2.302195872768437179e-01, 2.376123138654712419e-01,
        2.449786631268641435e-01, 2.596296294082575118e-01,
        2.741674511196587893e-01, 2.885873618940774099e-01,
        3.028848683749714166e-01, 3.170557532091470287e-01,
        3.310960767041321029e-01, 3.450021772071051318e-01,
        3.587706702705721895e-01, 3.723984466767542023e-01,
        3.858826693980737521e-01, 3.992207695752525431e-01,
        4.124104415973872673e-01, 4.254496373700422662e-01,
        4.383365598579578304e-01, 4.510696559885234436e-01,
        4.636476090008060935e-01, 4.883339510564055352e-01,
        5.123894603107377321e-01, 5.358112379604637043e-01,
        5.585993153435624414e-01, 5.807563535676704136e-01,
        6.022873461349641522e-01, 6.231993299340659043e-01,
        6.435011087932843710e-01, 6.632029927060932861e-01,
        6.823165548747480713e-01, 7.008544078844501923e-01,
        7.188299996216245269e-01, 7.362574289814280970e-01,
        7.531512809621944138e-01, 7.695264804056582975e-01,
        7.853981633974482790e-01, 8.156919233162234217e-01,
        8.441539861131710509e-01, 8.709034570756529758e-01,
        8.960553845713439269e-01, 9.197196053504168578e-01,
        9.420000403794636101e-01, 9.629943306809362058e-01,
        9.827937232473290541e-01, 1.001483135694234639e+00,
        1.019141344266349725e+00, 1.035841253008800145e+00,
        1.051650212548373764e+00, 1.066630365315743623e+00,
        1.080839000541168327e+00, 1.094328907321189925e+00,
        1.107148717794090409e+00, 1.130953743979160375e+00,
        1.152571997215667610e+00, 1.172273881128476303e+00,
        1.190289949682531656e+00, 1.206817370285252489e+00,
        1.222025323210989667e+00, 1.236059489478081863e+00,
        1.249045772398254428e+00, 1.261093382252440387e+00,
        1.272297395208717319e+00, 1.282740879744270757e+00,
        1.292496667789785336e+00, 1.301628834009196156e+00,
        1.310193935047555547e+00, 1.318242051016837113e+00,
        1.325817663668032553e+00, 1.339705659598999565e+00,
        1.352127380920954636e+00, 1.363300100359693845e+00,
        1.373400766945015894e+00, 1.382574821490125894e+00,
        1.390942827002418447e+00, 1.398605512271957618e+00,
        1.405647649380269870e+00, 1.412141064608495311e+00,
        1.418146998399631542e+00, 1.423717971406494032e+00,
        1.428899272190732761e+00, 1.433730152484709031e+00,
        1.438244794498222623e+00, 1.442473099109101931e+00,
        1.446441332248135092e+00,
 1.243549945467614382e-01, 1.320397616146387620e-01, 1.397088742891636204e-01,
 1.473614810886516302e-01, 1.549967419239409727e-01, 1.626138285979485676e-01,
 1.702119252854744080e-01, 1.777902289926760471e-01, 1.853479499956947607e-01,
 1.928843122579746439e-01, 2.003985538258785115e-01, 2.078899272022629863e-01,
 2.153576996977380476e-01, 2.228011537593945213e-01, 2.302195872768437179e-01,
 2.376123138654712419e-01, 2.449786631268641435e-01, 2.596296294082575118e-01,
 2.741674511196587893e-01, 2.885873618940774099e-01, 3.028848683749714166e-01,
 3.170557532091470287e-01, 3.310960767041321029e-01, 3.450021772071051318e-01,
 3.587706702705721895e-01, 3.723984466767542023e-01, 3.858826693980737521e-01,
 3.992207695752525431e-01, 4.124104415973872673e-01, 4.254496373700422662e-01,
 4.383365598579578304e-01, 4.510696559885234436e-01, 4.636476090008060935e-01,
 4.883339510564055352e-01, 5.123894603107377321e-01, 5.358112379604637043e-01,
```



```
 5.585993153435624414e-01, 5.807563535676704136e-01, 6.022873461349641522e-01,
 6.231993299340659043e-01, 6.435011087932843710e-01, 6.632029927060932861e-01,
 6.823165548747480713e-01, 7.008544078844501923e-01, 7.188299996216245269e-01,
 7.362574289814280970e-01, 7.531512809621944138e-01, 7.695264804056582975e-01,
 7.853981633974482790e-01, 8.156919233162234217e-01, 8.441539861131710509e-01,
 8.709034570756529758e-01, 8.960553845713439269e-01, 9.197196053504168578e-01,
 9.420000403794636101e-01, 9.629943306809362058e-01, 9.827937232473290541e-01,
 1.001483135694234639e+00, 1.019141344266349725e+00, 1.035841253008800145e+00,
 1.051650212548373764e+00, 1.066630365315743623e+00, 1.080839000541168327e+00,
 1.094328907321189925e+00, 1.107148717794090409e+00, 1.130953743979160375e+00,
 1.152571997215667610e+00, 1.172273881128476303e+00, 1.190289949682531656e+00,
 1.206817370285252489e+00, 1.222025323210989667e+00, 1.236059489478081863e+00,
 1.249045772398254428e+00, 1.261093382252440387e+00, 1.272297395208717319e+00,
 1.282740879744270757e+00, 1.292496667789785336e+00, 1.301628834009196156e+00,
 1.310193935047555547e+00, 1.318242051016837113e+00, 1.325817663668032553e+00,
 1.339705659598999565e+00, 1.352127380920954636e+00, 1.363300100359693845e+00,
 1.373400766945015894e+00, 1.382574821490125894e+00, 1.390942827002418447e+00,
 1.398605512271957618e+00, 1.405647649380269870e+00, 1.412141064608495311e+00,
 1.418146998399631542e+00, 1.423717971406494032e+00, 1.428899272190732761e+00,
 1.433730152484709031e+00, 1.438244794498222623e+00, 1.442473099109101931e+00,
 1.446441332248135092e+00,
};

static const double TBL_atan_lo[] = {
        -3.125324142453938311e-18, -1.276925400709959526e-17,
        2.479758919089733066e-17, 5.409599147666297957e-18,
        9.585415594114323829e-18, 7.784470643106252464e-18,
        -3.541164079802125137e-18, 2.372599351477449041e-17,
        4.180692268843078977e-18, 2.034098543938166622e-17,
        3.139954287184449286e-18, 7.333160666520898500e-18,
        4.738160130078732886e-19, -5.498822172446843173e-18,
        1.231340452914270316e-17, 1.058231431371112987e-17,
        1.069875561873445139e-17, 1.923875492461530410e-17,
        8.261353575163771936e-18, -1.428369957377257085e-17,
        -1.101082790300136900e-17, -1.893928924292642146e-17,
        -7.952610375793798701e-18, -2.293880475557830393e-17,
        3.088733564861919217e-17, 1.961231150484565340e-17,
        2.378822732491940868e-17, 2.246598105617042065e-17,
        3.963462895355093301e-17, 2.331553074189288466e-17,
        -2.494277030626540909e-17, 3.280735600183735558e-17,
        2.269877745296168709e-17, -1.137323618932958456e-17,
        -2.546278147285580353e-17, -4.063795683482557497e-18,
        -5.455630548591626394e-18, -1.441464378193066908e-17,
        2.950430737228402307e-17, 2.672403885140095079e-17,
        1.583478505144428617e-17, -3.076054864429649001e-17,
        6.943223671560007740e-18, -1.987626234335816123e-17,
        -2.147838844445698302e-17, 3.473937648299456719e-17,
        -2.425693465918206812e-17, -3.704991905602721293e-17,
        3.061616997868383018e-17, -1.071456562778743077e-17,
        -4.841337011934916763e-17, -2.269823590747287052e-17,
        2.923876285774304890e-17, -4.057439412852767923e-17,
        5.460837485846687627e-17, -3.986660595210752445e-18,
        1.390331103123099845e-17, 9.438308023545392000e-17,
        1.000401886936679889e-17, 3.194313981784503706e-17,
        -9.650564731467513515e-17, -5.956589637160374564e-17,
        -1.567632251135907253e-17, -5.490676155022364226e-18,
        9.404471373566379412e-17, 7.123833804538446299e-17,
        -9.159738508900378819e-17, 8.385188614028674371e-17,
        7.683333629842068806e-17, 4.172467638861439118e-17,
        -2.979162864892849274e-17, 7.879752739459421280e-17,
        -2.196203799612310905e-18, 3.242139621534960503e-17,
        2.245875015034507026e-17, -9.283188754266129476e-18,
        -6.830804768926660334e-17, -1.236918499824626670e-17,
        8.745413734780278834e-17, -6.319394031144676258e-17,
        -8.824429373951136321e-17, -2.599011860304134377e-17,
        2.147674250751150961e-17, 1.093246171526936217e-16,
```

```
        -3.307710355769516504e-17, -3.561490438648230100e-17,
        -9.843712133488842595e-17, -2.324061182591627982e-17,
        -8.922630138234492386e-17, -9.573807110557223276e-17,
        -8.263883782511013632e-17, 8.721870922223967507e-17,
        -6.457134743238754385e-17, -4.396204466767636187e-17,
        -2.493019910264565554e-17, -1.105119435430315713e-16,
        9.211323971545051565e-17,
-3.125324142453938311e-18, -1.276925400709959526e-17, 2.479758919089733066e-17,
5.409599147666297957e-18, 9.585415594114323829e-18, 7.784470643106252464e-18,
-3.541164079802125137e-18, 2.372599351477449041e-17, 4.180692268843078977e-18,
2.034098543938166622e-17, 3.139954287184449286e-18, 7.333160666520898500e-18,
4.738160130078732886e-19, -5.498822172446843173e-18, 1.231340452914270316e-17,
1.058231431371112987e-17, 1.069875561873445139e-17, 1.923875492461530410e-17,
8.261353575163771936e-18, -1.428369957377257085e-17, -1.101082790300136900e-17,
-1.893928924292642146e-17, -7.952610375793798701e-18, -2.293880475557830393e-17,
3.088733564861919217e-17, 1.961231150484565340e-17, 2.378822732491940868e-17,
2.246598105617042065e-17, 3.963462895355093301e-17, 2.331553074189288466e-17,
-2.494277030626540909e-17, 3.280735600183735558e-17, 2.269877745296168709e-17,
-1.137323618932958456e-17, -2.546278147285580353e-17, -4.063795683482557497e-18,
-5.455630548591626394e-18, -1.441464378193066908e-17, 2.950430737228402307e-17,
2.672403885140095079e-17, 1.583478505144428617e-17, -3.076054864429649001e-17,
6.943223671560007740e-18, -1.987626234335816123e-17, -2.147838844445698302e-17,
3.473937648299456719e-17, -2.425693465918206812e-17, -3.704991905602721293e-17,
3.061616997868383018e-17, -1.071456562778743077e-17, -4.841337011934916763e-17,
-2.269823590747287052e-17, 2.923876285774304890e-17, -4.057439412852767923e-17,
5.460837485846687627e-17, -3.986660595210752445e-18, 1.390331103123099845e-17,
9.438308023545392000e-17, 1.000401886936679889e-17, 3.194313981784503706e-17,
-9.650564731467513515e-17, -5.956589637160374564e-17, -1.567632251135907253e-17,
-5.490676155022364226e-18, 9.404471373566379412e-17, 7.123833804538446299e-17,
-9.159738508900378819e-17, 8.385188614028674371e-17, 7.683333629842068806e-17,
4.172467638861439118e-17, -2.979162864892849274e-17, 7.879752739459421280e-17,
-2.196203799612310905e-18, 3.242139621534960503e-17, 2.245875015034507026e-17,
-9.283188754266129476e-18, -6.830804768926660334e-17, -1.236918499824626670e-17,
8.745413734780278834e-17, -6.319394031144676258e-17, -8.824429373951136321e-17,
-2.599011860304134377e-17, 2.147674250751150961e-17, 1.093246171526936217e-16,
-3.307710355769516504e-17, -3.561490438648230100e-17, -9.843712133488842595e-17,
-2.324061182591627982e-17, -8.922630138234492386e-17, -9.573807110557223276e-17,
-8.263883782511013632e-17, 8.721870922223967507e-17, -6.457134743238754385e-17,
-4.396204466767636187e-17, -2.493019910264565554e-17, -1.105119435430315713e-16,
9.211323971545051565e-17,
};

/*
 * mx_atan(x,err)
 * Table look-up algorithm
 * By K.C. Ng, March 9, 1989
 *
 * Algorithm.
 *
 * The algorithm is based on atan(x)=atan(y)+atan((x-y)/(1+x*y)).
 * We use poly1(x) to approximate atan(x) for x in [0,1/8] with
 * error (relative)
 *      |(atan(x)-poly1(x))/x|<= 2^-83.41
 *
 * and use poly2(x) to approximate atan(x) for x in [0,1/65] with
 * error
 *      |atan(x)-poly2(x)|<= 2^-86.8
 *
 * Here poly1 and poly2 are odd polynomial with the following form:
 *              x + x^3*(a1+x^2*(a2+...))
 *
 * (0). Purge off Inf and NaN and 0
 * (1). Reduce x to positive by atan(x) = -atan(-x).
 * (2). For x <= 1/8, use
 *      (2.1) if x < 2^(-prec/2), atan(x) =  x  with inexact flag raised
 *      (2.2) Otherwise
```



```
 *              atan(x) = poly1(x)
 * (3). For x >= 8 then (prec = 78)
 *      (3.1) if x >= 2^prec,     atan(x) = atan(inf) - pio2lo
 *      (3.2) if x >= 2^(prec/3), atan(x) = atan(inf) - 1/x
 *      (3.3) if x >  65,         atan(x) = atan(inf) - poly2(1/x)
 *      (3.4) Otherwise,          atan(x) = atan(inf) - poly1(1/x)
 *
 * (4). Now x is in (0.125, 8)
 *      Find y that match x to 4.5 bit after binary (easy).
 *      If iy is the high word of y, then
 *              single : j = (iy - 0x3e000000) >> 19
 *              double : j = (iy - 0x3fc00000) >> 16
 *              quad   : j = (iy - 0x3ffc0000) >> 12
 *
 *      Let s = (x-y)/(1+x*y). Then
 *      atan(x) = atan(y) + poly1(s)
 *              = _TBL_atan_hi[j] + (_TBL_atan_lo[j] + poly2(s) )
 *
 *      Note. |s| <= 1.5384615385e-02 = 1/65. Maxium occurs at x = 1.03125
 *
 */

#define P1              p[2]
#define P4              p[8]
#define P5              p[9]
#define P6              p[10]
#define P7              p[11]
#define P8              p[12]
#define P9              p[13]

#endif /* ! codereview */
static const double p[] = {
        1.0,
        0.0,
        -3.33333333333333314830e-01,    /* p1   = BFD55555 55555555 */
        -1.85030852238476921863e-17,    /* p1_l = BC755525 9783A49C */
        2.00000000000000011102e-01,     /* p2   = 3FC99999 9999999A */
        -1.27263196576150347368e-17,    /* p2_l = BC6D584B 0D874007 */
        -1.42857142857141405923e-01,    /* p3   = BFC24924 9249245E */
        -1.34258204847170493327e-17,    /* p3_l = BC6EF534 A112500D */
        1.11111111110486909803e-01,     /* p4   = 3FBC71C7 1C71176A */
        -9.09090907557387889470e-02,    /* p5   = BFB745D1 73B47A7D */
        7.69230541541713053189e-02,     /* p6   = 3FB3B13A B1E68DE6 */
        -6.66645815401964159097e-02,    /* p7   = BFB110EE 1584446A */
        5.87081768778560317279e-02,     /* p8   = 3FAE0EFF 87657733 */
        -4.90818147456113240690e-02,    /* p9   = BFA92140 6A524B5C */
};

#endif /* ! codereview */
#define Q1              q[2]
#define Q3              q[6]
#define Q4              q[7]
#define Q5              q[8]

#endif /* ! codereview */
static const double q[] = {
        1.0,
        0.0,
        -3.33333333333333314830e-01,    /* q1   = BFD55555 55555555 */
        -1.85022941571278638733e-17,            /* q1_l = BC7554E9 D20EFA66 */
        1.99999999999999927836e-01,             /* q2   = 3FC99999 99999997 */
        -1.28782564407438833398e-17,            /* q2_l = BC6DB1FB 17217417 */
        -1.42857142855492280642e-01,            /* q3   = BFC24924 92483C46 */
        1.11111097130183356096e-01,             /* q4   = 3FBC71C6 E06595CC */
        -9.08553303569109294013e-02,            /* q5   = BFB7424B 808CDA76 */
};
```

```
321 static const double one = 1.0,
322         pio2hi = 1.570796326794896558e+00,
323         pio2lo = 6.123233995736765886e-17;
188 static const double
189 one = 1.0,
190 pio2hi = 1.570796326794896558e+00,
191 pio2lo = 6.123233995736765886e-17;

325 static double
326 mx_atan(double x, double *err)
327 {
328         double y, z, r, s, t, w, s_h, s_l, x_h, x_l, zz[3], ee[2], z_h, z_l,
329             r_h, r_l, u, v;
194 mx_atan(double x, double *err) {
195         double y, z, r, s, t, w, s_h, s_l, x_h, x_l, zz[3], ee[2], z_h,
196                 z_l, r_h, r_l, u, v;
330         int ix, iy, sign, j;

332         ix = ((int *)&x)[HIWORD];
199         ix = ((int *) &x)[HIWORD];
333         sign = ix & 0x80000000;
334         ix ^= sign;

336         /* for |x| < 1/8 */
337         if (ix < 0x3fc00000) {
338                 if (ix < 0x3f300000) {          /* when |x| < 2**-12 */
339                         if (ix < 0x3d800000) {  /* if |x| < 2**-39 */
340                                 *err = (double)((int)x);
207                                 *err = (double) ((int) x);
341                                 return (x);
342                         }

344 #endif /* ! codereview */
345                         z = x * x;
346                         t = x * z * (q[2] + z * (q[4] + z * q[6]));
347                         r = x + t;
348                         *err = t - (r - x);
349                         return (r);
350                 }

352 #endif /* ! codereview */
353                 z = x * x;

355                 /* use double precision at p4 and on */
356                 ee[0] = z * (P4 + z * (P5 + z * (P6 + z * (P7 + z * (P8 + z *
357                     P9)))));
210                 ee[0] = z *
211                         (P4 + z *
212                         (P5 + z * (P6 + z * (P7 + z * (P8 + z * P9)))));

359                 x_h = (double)((float)x);
360                 z_h = (double)((float)z);
214                 x_h = (double) ((float) x);
215                 z_h = (double) ((float) z);
361                 x_l = x - x_h;
362                 z_l = (x_h * x_h - z_h);
363                 zz[0] = z;
364                 zz[1] = z_h;
365                 zz[2] = z_l + x_l * (x + x_h);

367                 /*
368                  * compute (1+z*(p1+z*(p2+z*(p3+e)))) by call
369                  * mx_poly
370                  */
```



```
372                 mx_poly(zz, p, ee, 3);

374                 /* finally x*(1+z*(p1+...)) */
375                 r = x_h * ee[0];
376                 t = x * ee[1] + x_l * ee[0];
377                 s = t + r;
378                 *err = t - (s - r);
379                 return (s);
380         }

382 #endif /* ! codereview */
383         /* for |x| >= 8.0 */
384         if (ix >= 0x40200000) {                 /* x >=  8 */
385                 x = fabs(x);

387 #endif /* ! codereview */
388                 if (ix >= 0x42600000) {         /* x >=  2**39 */
389                         if (ix >= 0x44c00000)   /* x >=  2**77 */
236                         if (ix >= 0x44c00000) { /* x >=  2**77 */
390                                 y = -pio2lo;
391                         else
238                         } else
392                                 y = one / x - pio2lo;

394 #endif /* ! codereview */
395                         if (sign == 0) {
396                                 t = pio2hi - y;
397                                 *err = -(y - (pio2hi - t));
398                         } else {
399                                 t = y - pio2hi;
400                                 *err = y - (pio2hi + t);
401                         }

403 #endif /* ! codereview */
404                         return (t);
405                 } else {
406                         /* compute r = 1/x */
407                         r = one / x;
408                         z = r * r;

410 #endif /* ! codereview */
411                         if (ix < 0x40504000) {  /* 8 <  x <  65 */

412                                 /* use double precision at p4 and on */
413                                 ee[0] = z * (P4 + z * (P5 + z * (P6 + z * (P7 +
414                                     z * (P8 + z * P9)))));
415                                 x_h = (double)((float)x);
416                                 r_h = (double)((float)r);
417                                 z_h = (double)((float)z);
242                                 ee[0] = z *
243                                         (P4 + z *
244                                         (P5 + z *
245                                         (P6 + z * (P7 + z * (P8 + z * P9)))));
246                                 x_h = (double) ((float) x);
247                                 r_h = (double) ((float) r);
248                                 z_h = (double) ((float) z);
418                                 r_l = r * ((x_h - x) * r_h - (x_h * r_h - one));
419                                 z_l = (r_h * r_h - z_h);
420                                 zz[0] = z;
421                                 zz[1] = z_h;
422                                 zz[2] = z_l + r_l * (r + r_h);

424 #endif /* ! codereview */
425                                 /*
426                                  * compute (1+z*(p1+z*(p2+z*(p3+e)))) by call
427                                  * mx_poly
```

```
428                                  */
429                                 mx_poly(zz, p, ee, 3);
430                         } else {        /* x < 65 < 2**39 */
431                                 /* use double precision at q3 and on */
432                                 ee[0] = z * (Q3 + z * (Q4 + z * Q5));
433                                 x_h = (double)((float)x);
434                                 r_h = (double)((float)r);
435                                 z_h = (double)((float)z);
254                                 x_h = (double) ((float) x);
255                                 r_h = (double) ((float) r);
256                                 z_h = (double) ((float) z);
436                                 r_l = r * ((x_h - x) * r_h - (x_h * r_h - one));
437                                 z_l = (r_h * r_h - z_h);
438                                 zz[0] = z;
439                                 zz[1] = z_h;
440                                 zz[2] = z_l + r_l * (r + r_h);

442 #endif /* ! codereview */
443                                 /*
444                                  * compute (1+z*(q1+z*(q2+e))) by call
445                                  * mx_poly
446                                  */
447                                 mx_poly(zz, q, ee, 2);
448                         }

450 #endif /* ! codereview */
451                         /* pio2 - r*(1+...) */
452                         v = r_h * ee[0];
453                         t = pio2lo - (r * ee[1] + r_l * ee[0]);

455 #endif /* ! codereview */
456                         if (sign == 0) {
457                                 s = pio2hi - v;
458                                 t -= (v - (pio2hi - s));
459                         } else {
460                                 s = v - pio2hi;
461                                 t = -(t - (v - (s + pio2hi)));
462                         }

464 #endif /* ! codereview */
465                         w = s + t;
466                         *err = t - (w - s);
467                         return (w);
468                 }
469         }

471 #endif /* ! codereview */
472         /* now x is between 1/8 and 8 */
473         ((int *)&x)[HIWORD] = ix;
262         ((int *) &x)[HIWORD] = ix;
474         iy = (ix + 0x00008000) & 0x7fff0000;
475         ((int *)&y)[HIWORD] = iy;
476         ((int *)&y)[LOWORD] = 0;
264         ((int *) &y)[HIWORD] = iy;
265         ((int *) &y)[LOWORD] = 0;
477         j = (iy - 0x3fc00000) >> 16;

479         w = (x - y);
480         v = 1 / (one + x * y);
481         s = w * v;
482         z = s * s;
483         /* use double precision at q3 and on */
484         ee[0] = z * (Q3 + z * (Q4 + z * Q5));
485         s_h = (double)((float)s);
486         z_h = (double)((float)z);
487         x_h = (double)((float)x);
```



```
488         t = (double)((float)(one + x * y));
274         s_h = (double) ((float) s);
275         z_h = (double) ((float) z);
276         x_h = (double) ((float) x);
277         t = (double) ((float) (one + x * y));
489         r = -((x_h - x) * y - (x_h * y - (t - one)));
490         s_l = -v * (s_h * r - (w - s_h * t));
491         z_l = (s_h * s_h - z_h);
492         zz[0] = z;
493         zz[1] = z_h;
494         zz[2] = z_l + s_l * (s + s_h);
495         /* compute (1+z*(q1+z*(q2+e))) by call mx_poly */
496         mx_poly(zz, q, ee, 2);
497         v = s_h * ee[0];
498         t = TBL_atan_lo[j] + (s * ee[1] + s_l * ee[0]);
499         u = TBL_atan_hi[j];
500         s = u + v;
501         t += (v - (s - u));
502         w = s + t;
503         *err = t - (w - s);

505 #endif /* ! codereview */
506         if (sign != 0) {
507                 w = -w;
508                 *err = -*err;
509         }

511 #endif /* ! codereview */
512         return (w);
513 }

515 static const double twom768 = 6.441148769597133308e-232,        /* 2^-768 */
293 static const double
294         twom768 = 6.441148769597133308e-232,    /* 2^-768 */
516         two768 = 1.552518092300708935e+231,                     /* 2^768 */
517         pi = 3.141592653589793115997963468 5,
518         pi_lo = 1.224646799147353177e-16,
519         pio2 = 1.570796326794896558e+00,
520         pio2_lo = 6.123233995736765886e-17,
521         pio4 = 0.785398163397448278999 49,
522         pio4_lo = 3.061616997868382943e-17,
523         pi3o4 = 2.356194490192344836998,
524         pi3o4_lo = 9.184850993605148829195e-17;

526 double
527 __k_atan2(double y, double x, double *w)
528 {
306 __k_atan2(double y, double x, double *w) {
529         double t, xh, th, t1, t2, w1, w2;
530         int ix, iy, hx, hy, lx, ly;

532         hy = ((int *)&y)[HIWORD];
533         ly = ((int *)&y)[LOWORD];
310         hy = ((int *) &y)[HIWORD];
311         ly = ((int *) &y)[LOWORD];
534         iy = hy & ~0x80000000;

536         hx = ((int *)&x)[HIWORD];
537         lx = ((int *)&x)[LOWORD];
314         hx = ((int *) &x)[HIWORD];
315         lx = ((int *) &x)[LOWORD];
538         ix = hx & ~0x80000000;

540         *w = 0.0;

542 #endif /* ! codereview */
```

```
543         if (ix >= 0x7ff00000 || iy >= 0x7ff00000) {     /* ignore inexact */
544                 if (isnan(x) || isnan(y)) {
319                 if (isnan(x) || isnan(y))
545                         return (x * y);
546                 } else if (iy < 0x7ff00000) {
321                 else if (iy < 0x7ff00000) {
547                         if (hx >= 0) {  /* ATAN2(+-finite, +inf) is +-0 */
548                                 *w *= y;
549                                 return (*w);
550                         } else {        /* ATAN2(+-finite, -inf) is +-pi */
551                                 *w = copysign(pi_lo, y);
552                                 return (copysign(pi, y));
553                         }
554                 } else if (ix < 0x7ff00000) {
555                         /* ATAN2(+-inf, finite) is +-pi/2 */
556                         *w = (hy >= 0) ? pio2_lo : -pio2_lo;
557                         return ((hy >= 0) ? pio2 : -pio2);
331                         *w = (hy >= 0)? pio2_lo : -pio2_lo;
332                         return ((hy >= 0)? pio2 : -pio2);
558                 } else if (hx > 0) {    /* ATAN2(+-INF,+INF) = +-pi/4 */
559                         *w = (hy >= 0) ? pio4_lo : -pio4_lo;
560                         return ((hy >= 0) ? pio4 : -pio4);
334                         *w = (hy >= 0)? pio4_lo : -pio4_lo;
335                         return ((hy >= 0)? pio4 : -pio4);
561                 } else {                /* ATAN2(+-INF,-INF) = +-3pi/4 */
562                         *w = (hy >= 0) ? pi3o4_lo : -pi3o4_lo;
563                         return ((hy >= 0) ? pi3o4 : -pi3o4);
337                         *w = (hy >= 0)? pi3o4_lo : -pi3o4_lo;
338                         return ((hy >= 0)? pi3o4 : -pi3o4);
564                 }
565         } else if ((ix | lx) == 0 || (iy | ly) == 0) {
566                 if ((iy | ly) == 0) {
567                         if (hx >= 0) { /* ATAN2(+-0, +(0 <= x <= inf)) is +-0 */
342                         if (hx >= 0) /* ATAN2(+-0, +(0 <= x <= inf)) is +-0 */
568                                 return (y);
569                         } else { /* ATAN2(+-0, -(0 <= x <= inf)) is +-pi */
570                                 *w = (hy >= 0) ? pi_lo : -pi_lo;
571                                 return ((hy >= 0) ? pi : -pi);
344                         else {  /* ATAN2(+-0, -(0 <= x <= inf)) is +-pi */
345                                 *w = (hy >= 0)? pi_lo : -pi_lo;
346                                 return ((hy >= 0)? pi : -pi);
572                         }
573                 } else {  /* ATAN2(+-(anything but 0 and NaN), 0) is +-pi/2 */
574                         *w = (hy >= 0) ? pio2_lo : -pio2_lo;
575                         return ((hy >= 0) ? pio2 : -pio2);
349                         *w = (hy >= 0)? pio2_lo : -pio2_lo;
350                         return ((hy >= 0)? pio2 : -pio2);
576                 }
577         } else if (iy - ix > 0x06400000) {      /* |x/y| < 2 ** -100 */
578                 *w = (hy >= 0) ? pio2_lo : -pio2_lo;
579                 return ((hy >= 0) ? pio2 : -pio2);
353                 *w = (hy >= 0)? pio2_lo : -pio2_lo;
354                 return ((hy >= 0)? pio2 : -pio2);
580         } else if (ix - iy > 0x06400000) {      /* |y/x| < 2 ** -100 */
581                 if (hx < 0) {
582                         *w = (hy >= 0) ? pi_lo : -pi_lo;
583                         return ((hy >= 0) ? pi : -pi);
357                         *w = (hy >= 0)? pi_lo : -pi_lo;
358                         return ((hy >= 0)? pi : -pi);
584                 } else {
585                         t = y / x;
586                         th = t;
587                         ((int *)&th)[LOWORD] &= 0xf8000000;
362                         ((int *) &th)[LOWORD] &= 0xf8000000;
588                         xh = x;
589                         ((int *)&xh)[LOWORD] &= 0xf8000000;
```



```
364                         ((int *) &xh)[LOWORD] &= 0xf8000000;
590                         t1 = (x - xh) * t + xh * (t - th);
591                         t2 = y - xh * th;
592                         *w = (t2 - t1) / x;
593                         return (t);
594                 }
595         } else {
596                 if (ix >= 0x5f300000) {
597                         x *= twom768;
598                         y *= twom768;
599                 } else if (ix < 0x23d00000) {
600                         x *= two768;
601                         y *= two768;
602                 }

604 #endif /* ! codereview */
605                 y = fabs(y);
606                 x = fabs(x);
607                 t = y / x;
608                 th = t;
609                 ((int *)&th)[LOWORD] &= 0xf8000000;
378                 ((int *) &th)[LOWORD] &= 0xf8000000;
610                 xh = x;
611                 ((int *)&xh)[LOWORD] &= 0xf8000000;
380                 ((int *) &xh)[LOWORD] &= 0xf8000000;
612                 t1 = (x - xh) * t + xh * (t - th);
613                 t2 = y - xh * th;
614                 w1 = mx_atan(t, &w2);
615                 w2 += (t2 - t1) / (x + y * t);

617 #endif /* ! codereview */
618                 if (hx < 0) {
619                         t1 = pi - w1;
620                         t2 = pi - t1;
621                         w2 = (pi_lo - w2) - (w1 - t2);
622                         w1 = t1;
623                 }

625                 *w = (hy >= 0) ? w2 : -w2;
626                 return ((hy >= 0) ? w1 : -w1);
385                 *w = (hy >= 0)? w2 : -w2;
386                 return ((hy >= 0)? w1 : -w1);
627         }
628 }
```

*_____unchanged_portion_omitted_*

```
**********************************************************
   32535 Thu Jun  6 20:40:01 2019
new/usr/src/lib/libm/common/complex/k_atan2l.c
11210 libm should be cstyle(10NBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

#endif /* ! codereview */
/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2005 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#include "libm.h"                       /* __k_atan2l */
#include "complex_wrapper.h"

#if defined(__sparc)
#define HALF(x)         ((int *)&x)[3] = 0; ((int *)&x)[2] &= 0xfe000000
#define HALF(x)  ((int *) &x)[3] = 0; ((int *) &x)[2] &= 0xfe000000
#elif defined(__x86)
#define HALF(x)         ((int *)&x)[0] = 0
#define HALF(x)  ((int *) &x)[0] = 0
#endif

/*
 * long double __k_atan2l(long double y, long double x, long double *e)
 *
 * Compute atan2l with error terms.
 *
 * Important formula:
 *                  3       5
 *                 x       x
 * atan(x) = x - ----- + ----- - ...    (for x <= 1)
 *                 3       5
 *
 *           pi     1     1
 *         = --- - --- + --- - ...      (for x > 1)
 *                     3
 *            2     x    3x
 *
 * Arg(x + y i) = sign(y) * atan2(|y|, x)
 *              = sign(y) * atan(|y|/x)           (for x > 0)
```



```
 *                sign(y) * (PI - atan(|y|/|x|))  (for x < 0)
 * Thus if x >> y (IEEE double: EXP(x) - EXP(y) >= 60):
 *      1. (x > 0): atan2(y,x) ~ y/x
 *      2. (x < 0): atan2(y,x) ~ sign(y) (PI - |y/x|))
 * Otherwise if x << y:
 *      atan2(y,x) ~ sign(y)*PI/2 - x/y
 *
 * __k_atan2l call static functions mx_polyl, mx_atanl
 */

/*
 * (void) mx_polyl (long double *z, long double *a, long double *e, int n)
 * return
 *      e = a  + z*(a  + z*(a  + ... z*(a  + e)...))
 *           0       2       4           2n
 * Note:
 * 1.   e and coefficient ai are represented by two long double numbers.
 *      For e, the first one contain the leading 53 bits (30 for x86 exteneded)
 *      and the second one contain the remaining 113 bits (64 for x86 extended).
 *      For ai, the first one contian the leading 53 bits (or 30 for x86)
 *      rounded, and the second is the remaining 113 bits (or 64 for x86).
 * 2.   z is an array of three doubles.
 *      z[0] :  the rounded value of Z (the intended value of z)
 *      z[1] :  the leading 32 (or 56) bits of Z rounded
 *      z[2] :  the remaining 113 (or 64) bits of Z
 *              Note that z[0] = z[1]+z[2] rounded.
 *
 */

static void
mx_polyl(const long double *z, const long double *a, long double *e, int n)
{
mx_polyl(const long double *z, const long double *a, long double *e, int n) {
        long double r, s, t, p_h, p_l, z_h, z_l, p, w;
        int i;

#endif /* ! codereview */
        n = n + n;
        p = e[0] + a[n];
        p_l = a[n + 1];
        w = p;
        HALF(w);
        w = p; HALF(w);
        p_h = w;
        p = a[n - 2] + z[0] * p;
        z_h = z[1];
        z_l = z[2];
        z_h = z[1]; z_l = z[2];
        p_l += e[0] - (p_h - a[n]);

        for (i = n - 2; i >= 2; i -= 2) {

                /* compute p = ai + z * p */
                t = z_h * p_h;
                s = z[0] * p_l + p_h * z_l;
                w = p;
                HALF(w);
                w = p;  HALF(w);
                p_h = w;
                s += a[i + 1];
                r = t - (p_h - a[i]);
                p = a[i - 2] + z[0] * p;
                p_l = r + s;
        }
```

```
        w = p;
        HALF(w);
        w = p; HALF(w);
        e[0] = w;
        t = z_h * p_h;
        s = z[0] * p_l + p_h * z_l;
        r = t - (e[0] - a[0]);
        e[1] = r + s;
}

/*
 * Table of constants for atan from 0.125 to 8
 *      0.125 -- 0x3ffc0000  --- (increment at bit 12)
 *               0x3ffc1000
 *               0x3ffc2000
 *      ...     ...
 *               0x4001f000
 *      8.000 -- 0x40020000      (total: 97)
 */

static const long double TBL_atan_hil[] = {
#if defined(__sparc)
        1.2435499454676143503135484916387102416568e-01L,
        1.3203976161463874927468440652656953226250e-01L,
        1.3970887428916364518336777673909505681607e-01L,
        1.4736148108865163560980276039684551821066e-01L,
        1.5499674192394098230371437493349219133371e-01L,
        1.6261382859794857537364156376155780062019e-01L,
        1.7021192528547440449049660709976171369543e-01L,
        1.7779022899267607079662479921582468899456e-01L,
        1.8534794999569476488602596122854464667261e-01L,
        1.9288431225797466419705871069022730349878e-01L,
        2.0039855382587851465394578503437838446153e-01L,
        2.0788992720226299360533498310299432475629e-01L,
        2.1535769969773804802445962716648964165745e-01L,
        2.2280115375939451577103212214043255525024e-01L,
        2.3021958727684373024017095967980299065551e-01L,
        2.3761231386547125247388363432563777919892e-01L,
        2.4497866312686415417208248121127580641959e-01L,
        2.5962962940825753102994644318397190560106e-01L,
        2.7416745111965879759937189834217578592444e-01L,
        2.8858736189407739562361141995821834504332e-01L,
        2.0288486837497140556056094505582181227 7e-01L,
        3.1705575320914700980901557667446732975852e-01L,
        3.3109607670413209494433878775694455421259e-01L,
        3.4500217720710510886768128690005168408290e-01L,
        3.5877067027057222039592006392646052215363e-01L,
        3.7239844667675422192365503828370182641413e-01L,
        3.8588266939807377589769548460723139638186e-01L,
        3.9922076957525265614716696158864764911 04e-01L,
        4.1241044159738730689979128966712694260920e-01L,
        4.2544963737004228954226360518079233013817e-01L,
        4.3833655985795780544561604921477130895882e-01L,
        4.5106965598852347637563925728219344073798e-01L,
        4.6364760900080611621425623146121439713344e-01L,
        4.8833395105640552386716496074706484459644e-01L,
        5.1238946031073770666660102058425923805558e-01L,
        5.3581123796046370026908506870769144698471e-01L,
        5.5859931534356243597150821640166122875873e-01L,
        5.8075635356767039920327447500150082375122e-01L,
        6.0228734613496418168212269420423291922459e-01L,
        6.2319932993406593099247534906037459367793e-01L,
        6.4350110879328438680280922871732260447265e-01L,
        6.6320299270609325536325431023827583417226e-01L,
        6.8231655487474807825642998171115298784729e-01L,
        7.0085440788445017245795128178675127318623e-01L,
```



```
        7.1882999962162450541701415152590469891043e-01L,
        7.3625742898142813174283527108914662479274e-01L,
        7.5315128096219438952473937026902888600575e-01L,
        7.6952648040565826040682003598565401726598e-01L,
        7.8539816339744830961566084581987569936977e-01L,
        8.1569192331622341102146083874564582672284e-01L,
        8.4415398611317100251784414827164746738632e-01L,
        8.7090345707565295314017311259781407291650e-01L,
        8.9605538457134395617480071802993779546602e-01L,
        9.1971960535041681722860345482108940969311e-01L,
        9.4200004037946366473793717053459362115891e-01L,
        9.6299433068093620181519583599709989677298e-01L,
        9.8279372247329067985710611014666037625 72e-01L,
        1.0014831356942347329183295953014374896343e+00L,
        1.0191413442663497346383429170230636212354e+00L,
        1.0358412530088001765846944703254440735476e+00L,
        1.0516502125483736674598673120862999026920e+00L,
        1.0666303653157435630791763474202799086015e+00L,
        1.0808390005411683108871567292171997859003e+00L,
        1.0943289073211899198927883146102352763033e+00L,
        1.1071487177940905030170654601785370497543e+00L,
        1.1309537439791604464709335155363277560026e+00L,
        1.1525719972156675180401498626127514672834e+00L,
        1.1722738811284763866005949441337046006865e+00L,
        1.1902899496825317329277337748293182803384e+00L,
        1.2068173702852525303955115800565576625682e+00L,
        1.2220253232109896370417417439225704120294e+00L,
        1.2360594894780819419094519711090786146210e+00L,
        1.2490457723982544258299170772810900483550e+00L,
        1.2610933822524404193139408812473357640124e+00L,
        1.2722973952087173412961937498224805746463e+00L,
        1.2827408797442707473628852511364955164072e+00L,
        1.2924966677897852679030914214070816723528e+00L,
        1.3016288340091961438047858503666855024453e+00L,
        1.3101939350475556342564376891719053437537e+00L,
        1.3182420510168370498593302023271363040427e+00L,
        1.3258176636680324650592392104284756886164e+00L,
        1.3397056595989995393283037525895557850243e+00L,
        1.3521273809209546571891479413898127598774e+00L,
        1.3633001003596939542892985278250991560269e+00L,
        1.3734007669450158608612719264449610604836e+00L,
        1.3825748214901258580599674177685685163955e+00L,
        1.3909428270024183484267686943836432395486e+00L,
        1.3986055122719575950126700816114282727858e+00L,
        1.4056476493802697809521934019958080664406e+00L,
        1.4121410646084952153676136718584890852820e+00L,
        1.4181469983996314594038603039700988632607e+00L,
        1.4237179714064941189018190466107297108905e+00L,
        1.4288992721907326964184700745371984001389e+00L,
        1.4337301524847089866404719096698873880264e+00L,
        1.4382447944982225979614042479354816039669e+00L,
        1.4424730991091018200252920599377291810352e+00L,
        1.4464413322481351841999668424758803866109e+00L,
1.2435499454676143503135484916387102416568e-01L,
1.3203976161463874927468440652656953226250e-01L,
1.3970887428916364518336777673909505681607e-01L,
1.4736148108865163560980276039684551821066e-01L,
1.5499674192394098230371437493349219133371e-01L,
1.6261382859794857537364156376155780062019e-01L,
1.7021192528547440449049660709976171369543e-01L,
1.7779022899267607079662479921582468899456e-01L,
1.8534794999569476488602596122854464667261e-01L,
1.9288431225797466419705871069022730349878e-01L,
2.0039855382587851465394578503437838446153e-01L,
2.0788992720226299360533498310299432475629e-01L,
2.1535769969773804802445962716648964165745e-01L,
```

```
2.2280115375939451577103212214043255525024e-01L,
2.3021958727684373024017095967980299065551e-01L,
2.3761231386547125247388363432563777919892e-01L,
2.4497866312686415417208248121127580641959e-01L,
2.5962962940825753102994644318397190560106e-01L,
2.7416745111965879759937189834217578592444e-01L,
2.8858736189407739562361141995821834504332e-01L,
3.0288486837497140556055609450555821812277e-01L,
3.1705575320914700980901557667446732975852e-01L,
3.3109607670413209494433878775694455421259e-01L,
3.4500217720710510886768128690005168408290e-01L,
3.5877067027057222039592006392646052215363e-01L,
3.7239844667675422192365503828370182641413e-01L,
3.8588266939807377589769548460723139638186e-01L,
3.9922076957525256561471669615886476491104e-01L,
4.1241044159738730689979128966712694260920e-01L,
4.2544963737004228954226360518079233013817e-01L,
4.3833655985795780544561604921477130895882e-01L,
4.5106965598852347637563925728219344073798e-01L,
4.6364760900080611621425623146121439713344e-01L,
4.8833395105640552386716496074706484459644e-01L,
5.1238946031073770666660102058425923805558e-01L,
5.3581123796046370026908506870769144698471e-01L,
5.5859931534356243597150821640166122875873e-01L,
5.8075635356767039920327447500150082375122e-01L,
6.0228734613496418168212269420423291922459e-01L,
6.2319932993406593099247534906037459367793e-01L,
6.4350110879328438680280922871732260447265e-01L,
6.6320299270609325536325431023827583417226e-01L,
6.8231655487474807825642998171115298784729e-01L,
7.0085440788445017245795128178675127318623e-01L,
7.1882999962162450541701415152590469891043e-01L,
7.3625742898142813174283527108914662479274e-01L,
7.5315128096219438952473937026902888600575e-01L,
7.6952648040565826040682003598565401726598e-01L,
7.8539816339744830961566084581987569936977e-01L,
8.1569192331622341102146083874564582672284e-01L,
8.4415398611317100251784414827164746738632e-01L,
8.7090345707565295314017311259781407291650e-01L,
8.9605538457134395617480071802993779546602e-01L,
9.1971960535041681722860345482108940969311e-01L,
9.4200004037946366473793717053459362115891e-01L,
9.6299433068093620181519583599709989677298e-01L,
9.8279372247329067985710611014660376257 2e-01L,
1.0014831356942347329183295953014374896343e+00L,
1.0191413442663497346383429170230636212354e+00L,
1.0358412530088001765846944703254440735476e+00L,
1.0516502125483736674598673120862999026920e+00L,
1.0666303653157435630791763474202799086015e+00L,
1.0808390005411683108871567292171997859003e+00L,
1.0943289073211899198927883146102352763033e+00L,
1.1071487177940905030170654601785370497543e+00L,
1.1309537439791604464709335155363277560026e+00L,
1.1525719972156675180401498626127514672834e+00L,
1.1722738811284763866005949441337046006865e+00L,
1.1902899496825317329277337748293182803384e+00L,
1.2068173702852525303955115800565576625682e+00L,
1.2220253232109896370417417439225704120294e+00L,
1.2360594894780819419094519711090786146210e+00L,
1.2490457723982544258299170772810900483550e+00L,
1.2610933822524404193139408812473357640124e+00L,
1.2722973952087173412961937498224805746463e+00L,
1.2827408797442707473628852511364955164072e+00L,
1.2924966677897852679030914214070816723528e+00L,
1.3016288340091961438047858503666855024453e+00L,
1.3101939350475556342564376891719053437537e+00L,
```



```
1.3182420510168370498593302023271363040427e+00L,
1.3258176636680324650592392104284756886164e+00L,
1.3397056595989995393283037525895557850243e+00L,
1.3521273809209546571891479413898127598774e+00L,
1.3633001003596939542892985278250991560269e+00L,
1.3734007669450158608612719264449610604836e+00L,
1.3825748214901258580599674177685685163955e+00L,
1.3909428270024183486427686943836432395486e+00L,
1.3986055122719575950126700816114282727858e+00L,
1.4056476493802697809521934019958080664406e+00L,
1.4121410646084952153676136718584890852820e+00L,
1.4181469839963145940386030397009886326 07e+00L,
1.4237179714064941189018190466107297108905e+00L,
1.4288992721907326964184700745371984001389e+00L,
1.4337301524847089866404719096698873880264e+00L,
1.4382447944982225979614042479354816039669e+00L,
1.4424730991091018200252920599377291810352e+00L,
1.4464413322481351841999668424758803866109e+00L,
#elif defined(__x86)
        1.2435499453567899763584136962 89e-01L,
        1.3203976157819852232933044435 59e-01L,
        1.3970887428149580955505371093 75e-01L,
        1.4736148103838786482810974121 09e-01L,
        1.5499674191232770681381225585 94e-01L,
        1.6261382855009287595748901367 19e-01L,
        1.7021192522952333092689514160 16e-01L,
        1.7779022897593677043914794921 88e-01L,
        1.8534749496958225965499877929 69e-01L,
        1.9288431224413216114044189453 12e-01L,
        2.0039855380309745669364929199 22e-01L,
        2.0788992720190435647964477539 06e-01L,
        2.1535769966430962085723876953 12e-01L,
        2.2280115372268483042716979980 47e-01L,
        2.3021958727622404694557189941 41e-01L,
        2.3761231382377445697784423828 12e-01L,
        2.4497866310412064194679260253 91e-01L,
        2.5962962931953370571136474609 38e-01L,
        2.7416745107620954513549804687 50e-01L,
        2.8858736180700361728668212890 62e-01L,
        3.0288486834615468978881835937 50e-01L,
        3.1705575319938361644744873046 88e-01L,
        3.3109607663936913013458251953 12e-01L,
        3.4500217717140913009643554687 50e-01L,
        3.5877067025285214185714721679 69e-01L,
        3.7239844666328281164169311523 44e-01L,
        3.8588266936130821704864501953 12e-01L,
        3.9922076952643692493438720703 12e-01L,
        4.1241044155322015285491943359 38e-01L,
        4.2544963734690099954605102539 06e-01L,
        4.3833655980415642261505126953 12e-01L,
        4.5106965594459325075149536132 81e-01L,
        4.6364760899450629949569702148 44e-01L,
        4.8833395098336040973663330078 12e-01L,
        5.1238946011289954185485839843 75e-01L,
        5.3581123775802552700042724609 38e-01L,
        5.5859931511804461479187011718 75e-01L,
        5.8075635344721376895904541015 62e-01L,
        6.0228734603151679039001464843 75e-01L,
        6.2319932971149682998657226562 50e-01L,
        6.4350110874511301517486572265 62e-01L,
        6.6320299264043569564819335937 50e-01L,
        6.8231655471026897430419921875 00e-01L,
        7.0085440785624086856842041015 62e-01L,
        7.1882999944500625133514404296 88e-01L,
        7.3625742876902222633361816406 25e-01L,
        7.5315128080546855926513671875 00e-01L,
```

```
        7.695264802314341068267822265625e-01L,
        7.853981633670628070831298828125e-01L,
        8.156919232569634914398193359375e-01L,
        8.441539860796183347702026367188e-01L,
        8.709034570492804050445556640625e-01L,
        8.960553845390677452087402343750e-01L,
        9.197196052409708499908447265625e-01L,
        9.420000403188169002532958984375e-01L,
        9.629943305626511573791503906250e-01L,
        9.827937232330441474914550781250e-01L,
        1.001483135391026735305786132812e+00L,
        1.051650212146341800689697265625e+00L,
        1.066630364861339330673217773438e+00L,
        1.080839000176638364791870117188e+00L,
        1.094328907318413257598876953125e+00L,
        1.107148717623203992843627929688e+00L,
        1.130953743588179349899291992188e+00L,
        1.152571997139602899551391601562e+00L,
        1.172273880802094936370849609375e+00L,
        1.190289949532598257064819335938e+00L,
        1.206817369908094406127929687500e+00L,
        1.222025323193520307540893554688e+00L,
        1.236059489194303750991821289062e+00L,
        1.249045772012323141098022460938e+00L,
        1.261093381792306900024414062500e+00L,
        1.272297394927591085433959960938e+00L,
        1.282740879338234663009643554688e+00L,
        1.292496667709201574325561523438e+00L,
        1.301628833636641502380371093750e+00L,
        1.310193934943526983261108398438e+00L,
        1.318242050707340240478515625000e+00L,
        1.325817663222551345825195312500e+00L,
        1.339705659542232751846313476562e+00L,
        1.352127380669116973876953125000e+00L,
        1.363300099968910217285156250000e+00L,
        1.373400766868144273757934570312e+00L,
        1.382574821356683969497680664062e+00L,
        1.390942826867103576660156250000e+00L,
        1.398605511989444494247436523438e+00L,
        1.405647648964077234268188476562e+00L,
        1.412141064181923866271972656250e+00L,
        1.418146998155862092971801757812e+00L,
        1.423717970959842205047607421875e+00L,
        1.428899271879345178604125976562e+00L,
        1.433730152435600757598876953125e+00L,
        1.438244794495403766632080078125e+00L,
        1.442473099101334810256958007812e+00L,
        1.446441331878304481506347656250e+00L,
1.243549945356789976358413696289e-01L,  1.320397615781985223293304443359e-01L,
1.397088742814958095550537109375e-01L,  1.473614810383878648281097412109e-01L,
1.549967419123277068138122558594e-01L,  1.626138285500928759574890136719e-01L,
1.702119252295233309268951416016e-01L,  1.777902289759367704391479492188e-01L,
1.853479499695822596549987792969e-01L,  1.928843122441321611404418945312e-01L,
2.003985538030974566936492919922e-01L,  2.078899272019043564796447753906e-01L,
2.153576996643096208572387695312e-01L,  2.228011537226848304271697998047e-01L,
2.302195872762240469455718994141e-01L,  2.376123138237744569778442382812e-01L,
2.449786631041206419467926025391e-01L,  2.596296293195337057113647460938e-01L,
2.741674510762095451354980468750e-01L,  2.885873618070036172866821289062e-01L,
3.028848683461546897888183593750e-01L,  3.170557531993836164474487304688e-01L,
3.310960766393691301345825195312e-01L,  3.450021771714091300964355468750e-01L,
3.587706702585214185714721679969e-01L,  3.723984466632828116416931152344e-01L,
3.858826936130821704864501953125e-01L,  3.992207695264369249343872070312e-01L,
4.124104415532201528549194335938e-01L,  4.254496373469009995460510253906e-01L,
4.383365598041564226150512695312e-01L,  4.510696559445932507514953613281e-01L,
```



```
4.636476089945062994956970214844e-01L,  4.883339509833604097366333007812e-01L,
5.123894601128995418548583984375e-01L,  5.358112377580255270004272460938e-01L,
5.585993151180446147918701171875e-01L,  5.807563534472137689590454101562e-01L,
6.022873460315167903900146484375e-01L,  6.231993297114968299865722656250e-01L,
6.435011087451130151748657226562e-01L,  6.632029926404356956481933593750e-01L,
6.823165547102689743041992187500e-01L,  7.008544078562408685684204101562e-01L,
7.188299994450062513351440429688e-01L,  7.362574287690222263336181640625e-01L,
7.531512808054685592651367187500e-01L,  7.695264802314341068267822265625e-01L,
7.853981633670628070831298828125e-01L,  8.156919232569634914398193359375e-01L,
8.441539860796183347702026367188e-01L,  8.709034570492804050445556640625e-01L,
8.960553845390677452087402343750e-01L,  9.197196052409708499908447265625e-01L,
9.420000403188169002532958984375e-01L,  9.629943305626511573791503906250e-01L,
9.827937232330441474914550781250e-01L,  1.001483135391026735305786132812e+00L,
1.019141343887895345687866210938e+00L,  1.035841252654790878295898437500e+00L,
1.051650212146341800689697265625e+00L,  1.066630364861339330673217773438e+00L,
1.080839000176638364791870117188e+00L,  1.094328907318413257598876953125e+00L,
1.107148717623203992843627929688e+00L,  1.130953743588179349899291992188e+00L,
1.152571997139602899551391601562e+00L,  1.172273880802094936370849609375e+00L,
1.190289949532598257064819335938e+00L,  1.206817369908094406127929687500e+00L,
1.222025323193520307540893554688e+00L,  1.236059489194303750991821289062e+00L,
1.249045772012323141098022460938e+00L,  1.261093381792306900024414062500e+00L,
1.272297394927591085433959960938e+00L,  1.282740879338234663009643554688e+00L,
1.292496667709201574325561523438e+00L,  1.301628833636641502380371093750e+00L,
1.310193934943526983261108398438e+00L,  1.318242050707340240478515625000e+00L,
1.325817663222551345825195312500e+00L,  1.339705659542232751846313476562e+00L,
1.352127380669116973876953125000e+00L,  1.363300099968910217285156250000e+00L,
1.373400766868144273757934570312e+00L,  1.382574821356683969497680664062e+00L,
1.390942826867103576660156250000e+00L,  1.398605511989444494247436523438e+00L,
1.405647648964077234268188476562e+00L,  1.412141064181923866271972656250e+00L,
1.418146998155862092971801757812e+00L,  1.423717970959842205047607421875e+00L,
1.428899271879345178604125976562e+00L,  1.433730152435600757598876953125e+00L,
1.438244794495403766632080078125e+00L,  1.442473099101334810256958007812e+00L,
1.446441331878304481506347656250e+00L,
#endif
};
```

```
#endif /* ! codereview */
static const long double TBL_atan_lol[] = {
#if defined(__sparc)
        1.407486919762806380231720282041431003955 6e-36L,
        -4.959696159473992555573043943799967529550 5e-36L,
        8.952774562519464887393121344636184947278 8e-36L,
        1.188043742320789571818076584354496558942 7e-35L,
        -2.781027811204514537842537512823436538144 8e-37L,
        1.479722037702380032729553623431514726238 7e-36L,
        -4.216956140054819873287038480184963986382 9e-36L,
        7.243122966691348464993032365631602349468 0e-36L,
        -2.157343008983917029989567935379066318246 2e-36L,
        -9.951574540512672355445236729812860518630 5e-36L,
        -3.906555899232483818161756973039788236306 7e-36L,
        5.526029227179372681321198066466112451880 7e-36L,
        8.841572221591432180768225431803645204368 9e-36L,
        -8.176772879158617925419332362828559980071 1e-36L,
        -1.334412303465614224379711382302833007076 2e-36L,
        -4.492733120781338290893073392468132589218 8e-36L,
        4.494551147181249039320182433676249568773 0e-36L,
        -1.668808150427922355577672445964844056727 4e-35L,
        1.562975758610795576946108656893732968411 3e-35L,
        -2.238983556330807855250797038533151084810 9e-35L,
        -4.831232174554731155187045067118215136705 0e-36L,
        -1.433617235290583287659892661098069884430 9e-35L,
        -8.744018199889993280298917417096059331608 0e-36L,
        5.928463600852983744578036078546455014301 6e-36L,
        -2.237665124843624127606105529504351499363 0e-35L,
        6.074583759933610541428031075667744213648 0e-36L,
        1.537218711045194967779234476202996702309 3e-35L,
```

```
        2.0976068056751156241657121582478790247159e-35L,
        -5.5623956405495438060726862202622807523700e-36L,
        1.9697366707832471841858411934897351901523e-35L,
        2.1070311964479488509034733639424887543697e-35L,
        -2.3027356362982001602256518510854229844561e-35L,
        4.8950964225733349266861843522029764772843e-36L,
        -7.2380143477794458213872723050820253166391e-36L,
        1.6365648865703614031637443396049568858105e-35L,
        -3.9885811958234530793729129919803234197399e-35L,
        4.1587722120912613510417783923227421336929e-35L,
        3.8347421454556472153684687377337135027394e-35L,
        -9.2251178933638721723515896465489002497864e-36L,
        1.4094619690455989526175736741854656192178e-36L,
        3.3568857805472235270612851425810803679451e-35L,
        3.9090991055522552395018106803232118803401e-35L,
        5.2956416979654208140521862707297033857956e-36L,
        -5.0960846819945514367847063923662507136721e-36L,
        -4.4959014425277615858329680393918315204998e-35L,
        3.8039226544551634266566857615962609653834e-35L,
        -4.4056522872895512108308642196611689657618e-36L,
        1.6025024192482161076223807753425619076948e-36L,
        2.1679525325309452561992610065108380635264e-35L,
        1.9844038013515422125715362925736754104066e-35L,
        3.9139619471799746834505227353568432457241e-35L,
        2.1113443807975453505518453436799561854730e-35L,
        3.1558557277444692755039816944392770185432e-35L,
        1.6295044520355461408265585619500238335614e-35L,
        -3.5087245209270305856151230356171213582305e-35L,
        2.9041041864282855679591055270946117300088e-35L,
        -2.3128843453818356590931995209806627233282e-35L,
        -7.7124923181471578439967973820714857839953e-35L,
        2.7539027829886922429092063590445808781462e-35L,
        -9.4500899453181308951084545990839335972452e-35L,
        -7.3061755302032092337594946001641651543473e-35L,
        -4.1736144813953752193952770157406952602798e-35L,
        3.4369948356256407045344855262863733571105e-35L,
        -6.3790243492298090907302084924276831116460e-35L,
        -9.6842943816353261291004127866079538980649e-36L,
        4.8746757539138870909275958326700072821615e-35L,
        -8.7533886477084190884511601368582548254655e-35L,
        1.4284743992327918892692551138086727754845e-35L,
        5.7262776211073389542565625693479173445042e-35L,
        -3.2254883148780411245594822270747948565684e-35L,
        7.8853548190609877325965525252380833808405e-35L,
        8.4081736739037194097515038365370730251333e-35L,
        7.4722870357563683815078242981933587273670e-35L,
        7.9977202825793435289434813600890494256112e-36L,
        -8.0577840773362139054848492346292673645405e-35L,
        1.4217746753670583065490040209048757624336e-35L,
        1.2232486914221205004109743560319090913328e-35L,
        8.9696055070830036447361957217943988339065e-35L,
        -3.1480394435081884410686066739846298958951e-35L,
        -5.0927146040715345013240642517608928352977e-35L,
        -5.7431997715924136568133859432702789493569e-35L,
        -4.3920451405083770279099766080476485439987e-35L,
        9.1106753984907715563018666776308759323326e-35L,
        -3.7032569014272841009512400773061537538358e-35L,
        8.8167419429746714276909825405131416764489e-35L,
        -3.8389341696028352503752312861740895209678e-36L,
        -3.3462959341960891546340895508017603408404e-35L,
        -3.9212626776786074383916188498955828634947e-35L,
        -7.8340397396377867255864494568594088378648e-35L,
        7.4681018632456986520600640340627309824469e-35L,
        8.9110918618956918451135594876165314884113e-35L,
        3.9418160632271890530431797145664308529115e-35L,
        -4.1048114088580104820193435638327617443913e-35L,
```



```
        -2.3165419451582153326383944756220900454330e-35L,
        -1.8428312581525319409399330203703211113843e-35L,
        7.1477316546709482345411712017906842769961e-35L,
        2.9914501578435874662153637707016094237004e-35L,
1.4074869197628063802317202820414310039556e-36L,
-4.9596961594739925555730439437999675295505e-36L,
8.9527745625194648873931213446361849472788e-36L,
1.1880437423207895718180765843544965589427e-35L,
-2.7810278112045145378425375128234365381448e-37L,
1.4797220377023800327295536234315147262387e-36L,
-4.2169561400548198732870384801849639863829e-36L,
7.2431229666913484649930323656316023494680e-36L,
-2.1573430089839170299895679353790663182462e-36L,
-9.9515745405126723554452367298128605186305e-36L,
-3.9065558992324838181617569730397882363067e-36L,
5.5260292271793726813211980664661124518807e-36L,
8.8415722215914321807682254318036452043689e-36L,
-8.1767728791586179254193323628285599800711e-36L,
-1.3344123034656142243797113823028330070762e-36L,
-4.4927331207813382908930733924681325892188e-36L,
4.4945511471812490393201824336762495687730e-36L,
-1.6688081504279223555776724459648440567274e-35L,
1.5629757586107955769461086568937329684113e-35L,
-2.2389835563308078552507970385331510848109e-35L,
-4.8312321745547311551870450671182151367050e-36L,
-1.4336172352905832876958926610980698844309e-35L,
-8.7440181998899328029891741709605933160080e-36L,
5.9284636008529837445780360785464550143016e-36L,
-2.2376651248436241276061055295043514993630e-35L,
6.0745837599336105414280310756677442136480e-36L,
1.5372187110451949677792344762029967023093e-35L,
2.0976068056751156241657121582478790247159e-35L,
-5.5623956405495438060726862202622807523700e-36L,
1.9697366707832471841858411934897351901523e-35L,
2.1070311964479488509034733639424887543697e-35L,
-2.3027356362982001602256518510854229844561e-35L,
4.8950964225733349266861843522029764772843e-36L,
-7.2380143477794458213872723050820253166391e-36L,
1.6365648865703614031637443396049568858105e-35L,
-3.9885811958234530793729129919803234197399e-35L,
4.1587722120912613510417783923227421336929e-35L,
3.8347421454556472153684687377337135027394e-35L,
-9.2251178933638721723515896465489002497864e-36L,
1.4094619690455989526175736741854656192178e-36L,
3.3568857805472235270612851425810803679451e-35L,
3.9090991055522552395018106803232118803401e-35L,
5.2956416979654208140521862707297033857956e-36L,
-5.0960846819945514367847063923662507136721e-36L,
-4.4959014425277615858329680393918315204998e-35L,
3.8039226544551634266566857615962609653834e-35L,
-4.4056522872895512108308642196611689657618e-36L,
1.6025024192482161076223807753425619076948e-36L,
2.1679525325309452561992610065108380635264e-35L,
1.9844038013515422125715362925736754104066e-35L,
3.9139619471799746834505227353568432457241e-35L,
2.1113443807975453505518453436799561854730e-35L,
3.1558557277444692755039816944392770185432e-35L,
1.6295044520355461408265585619500238335614e-35L,
-3.5087245209270305856151230356171213582305e-35L,
2.9041041864282855679591055270946117300088e-35L,
-2.3128843453818356590931995209806627233282e-35L,
-7.7124923181471578439967973820714857839953e-35L,
2.7539027829886922429092063590445808781462e-35L,
-9.4500899453181308951084545990839335972452e-35L,
-7.3061755302032092337594946001641651543473e-35L,
-4.1736144813953752193952770157406952602798e-35L,
```

```
3.436994835625640704534485526286373357110 5e-35L,
-6.379024349229809090730208492427683111646 0e-35L,
-9.684294381635326129100412786607953898064 9e-36L,
4.874675753913887090927595832670007282161 5e-35L,
-8.753388647708419088451160136858254825465 5e-35L,
1.428474399232791889269255113808672775484 5e-35L,
5.726277621107338954256562569347917344504 2e-35L,
-3.225488314878041124559482227074794856568 4e-35L,
7.885354819060987732596552525238083380840 5e-35L,
8.408173673903719409751503836537073025133 3e-35L,
7.472287035756368381507824298193358727367 0e-35L,
7.997720282579343528943481360089049425611 2e-36L,
-8.057784077336213905484849234629267364540 5e-35L,
1.421774675367058306549004020904875762433 6e-35L,
1.223248691422120500410974356031909091332 8e-35L,
8.969605507083003644736195721794398833906 5e-35L,
-3.148039443508188441068606673984626985895 1e-35L,
-5.092714604071534501324064251760892835297 7e-35L,
-5.743199771592413656813385943270789493569 e-35L,
-4.392045140508377027909976608047648543998 7e-35L,
9.110675398490771556301866677630875932332 6e-35L,
-3.703256901427284100951240077306153753885 8e-35L,
8.816741942974671427690982540513141676448 9e-35L,
-3.838934169602835250375231286174089520967 8e-36L,
-3.346295934196089154634089550801760340840 4e-35L,
-3.921262677678607438391618849895582863494 7e-35L,
-7.834039739637786725586449456859408837864 8e-35L,
7.468101863245698652060064034062730982446 9e-35L,
8.911091861895691845113559487616531488411 3e-35L,
3.941816063227189053043179714566430852911 5e-35L,
-4.104811408858010482019343563832761744391 3e-35L,
-2.316541945158215332638394475622090045433 0e-35L,
-1.842831258152531940939933020370321111384 3e-35L,
7.147731654670948234541171201790684276996 1e-35L,
2.991450157843587466215363770701609423700 4e-35L,
#elif defined(__x86)
        1.108243739551347953496477557317e-11L,
        3.644022694535396219063202730280e-11L,
        7.667835628314065801595065768845e-12L,
        5.026377078169301918590803009109e-11L,
        1.161327548990211907411719105561e-11L,
        4.785569941615255008968280209991e-11L,
        5.595107356360146549819920947848e-11L,
        1.673930035747684999707469623769e-11L,
        2.611250523102718193166964451527e-11L,
        1.384250305661681615897729354721e-11L,
        2.278105796029649304219088055497e-11L,
        3.586371256902077123693302823191e-13L,
        3.342842716722085763523965049902e-11L,
        3.670968534386232233574504707347e-11L,
        6.196832945990602657404893210974e-13L,
        4.169679549603939604438777470618e-11L,
        2.274351222528987867221331091414e-11L,
        8.872382531858169709022188891298e-11L,
        4.344925246387385146717580155420e-11L,
        8.707377833692929105196832265348e-11L,
        2.881671577173773513055821329154e-11L,
        9.763393361566846205717315422347e-12L,
        6.476296480975626822569454546857e-11L,
        3.569597877124574002505169001136e-11L,
        1.772007853877284712958549977698e-11L,
        1.347141028196192304932683248872e-11L,
        3.676555884905046507598141175404e-11L,
        4.881564068032948912761478588710e-11L,
        4.416715404487185607337693704681e-11L,
        2.314128999621257979016734983553e-11L,
```



```
        5.380138283056477968352133002913e-11L,
        4.393022562414389595406841771063e-11L,
        6.299816718559209976839402028537e-12L,
        7.304511413053165996581483735843e-11L,
        1.978381648117426221467592544212e-10L,
        2.024381732686578226139414070989e-10L,
        2.255178211796380992141612703464e-10L,
        1.204566302442290648452508620986e-10L,
        1.034473912921080457667329099995e-10L,
        2.225691010059030834353745950874e-10L,
        4.817137162794350606107263804151e-11L,
        6.565755971506095086327587326326e-11L,
        1.644791039522307629611529931429e-10L,
        2.820930388953087163050126809014e-11L,
        1.766182540818701085571546539514e-10L,
        2.124059054092171070266466628320e-10L,
        1.567258302596026515190288816001e-10L,
        1.742241535800378094231540188685e-10L,
        3.038550253253096300737572104929e-11L,
        5.925991958164150280814584656688e-11L,
        3.355266774764151155289750652594e-11L,
        2.637254809561744853531409402995e-11L,
        3.227621096606048365493782702458e-11L,
        1.094459672377587282585894259882e-10L,
        6.064676448464127209709358607166e-11L,
        1.182850444360454453720999258140e-10L,
        1.428492049425553288966601449688e-11L,
        3.032079976125434624889374125094e-10L,
        3.784543889504767060855636487744e-10L,
        3.540092982887960328254439790467e-10L,
        4.020318667701700464612998296302e-10L,
        4.544042324059585739827798668654e-10L,
        3.645299460952866120296998202703e-10L,
        2.776662293911361485235212513020e-12L,
        1.708865101734375304910370400700e-10L,
        3.909810965716415233488278047493e-10L,
        7.606461848875826105025137974947e-11L,
        3.263814502297453347587046149712e-10L,
        1.499334758629144388918183376012e-10L,
        3.771581242675818925565576303133e-10L,
        1.746932950084818923507049088298e-11L,
        2.837781909176306820465786987027e-10L,
        3.859312847318946163435901230778e-10L,
        4.601335192895268187473357720101e-10L,
        2.811262558622337888849804940684e-10L,
        4.060360843532416964489955306249e-10L,
        8.058369357752989796958168458531e-11L,
        3.725546414244147566166855921414e-10L,
        1.040286509953292907344053122733e-10L,
        3.094968093808145773271362531155e-10L,
        4.454811192340438979284756311844e-10L,
        5.676678748199027602705574110388e-11L,
        2.518376833121948163898128509842e-10L,
        3.907837370041422778250991189943e-10L,
        7.687158710333735613246114865100e-11L,
        1.334418885622867537060685125566e-10L,
        1.353147719826124443836432060856e-10L,
        2.825131007652335581739282335732e-10L,
        4.161925466840049254333079881002e-10L,
        4.265713490956410156084891599630e-10L,
        2.437693664320585461575989523716e-10L,
        4.466519138542116247357297503086e-10L,
        3.113875178143440979746983590908e-10L,
        4.910822904159495654488736486097e-11L,
        2.818831329324169810481585538618e-12L,
        7.767009768334052125229252512543e-12L,
```

```
        3.698307026936191862258804165254e-10L,
1.108243739551347953496477557317e-11L,  3.644022694535396219063202730280e-11L,
7.667835628314065801595065768845e-12L,  5.026377078169301918590803009109e-11L,
1.161327548990211907411719105561e-11L,  4.785569941615255008968280209991e-11L,
5.595107356360146549819920947848e-11L,  1.673930035747684999707469623769e-11L,
2.611250523102718193166964451527e-11L,  1.384250305661681615897729354721e-11L,
2.278105796029649304219088055497e-11L,  3.586371256902077123693302823191e-13L,
3.342842716722085763523965049902e-11L,  3.670968534386232233574504707347e-11L,
6.196832945990602657404893210974e-13L,  4.169679549603939604438777470618e-11L,
2.274351222528987867221331091414e-11L,  8.872382531858169709022188891298e-11L,
4.344925246387385146717580155420e-11L,  8.707377833692929105196832265348e-11L,
2.881671577173773513055821329154e-11L,  9.763393361566846205717315422347e-12L,
6.476296480975626822569454546857e-11L,  3.569597877124574002505169001136e-11L,
1.772007853877284712958549977698e-11L,  1.347141028196192304932683248872e-11L,
3.676555884905046507598141175404e-11L,  4.881564068032948912761478588710e-11L,
4.416715404487185607337693704681e-11L,  2.314128999621257979016734983553e-11L,
5.380138283056477968352133002913e-11L,  4.393022562414389595406841771063e-11L,
6.299816718559209976839402028537e-12L,  7.304511413053165996581483735843e-11L,
1.978381648117426221467592544212e-10L,  2.024381732686578226139414070989e-10L,
2.255178211796380992141612703464e-10L,  1.204566302442290648452508620986e-10L,
1.034473912921080457667329099995e-10L,  2.225691010059030834353745950874e-10L,
4.817137162794350606107263804151e-11L,  6.565755971506095086327587326326e-11L,
1.644791039522307629611529931429e-10L,  2.820930388953087163050126809014e-11L,
1.766182540818701085571546539514e-10L,  2.124059054092171070266466628320e-10L,
1.567258302596026515190288816001e-10L,  1.742241535800378094231540188685e-10L,
3.038550253253096300737572104929e-11L,  5.925991958164150280814584656688e-11L,
3.355266774764151155289750652594e-11L,  2.637254809561744853531409402995e-11L,
3.227621096606048365493782702458e-11L,  1.094459672377587282585894259882e-10L,
6.064676448464127209709358607166e-11L,  1.182850444360454453720999258140e-10L,
1.428492049425553288966601449688e-11L,  3.032079976125434624889374125094e-10L,
3.784543889504767060855636487744e-10L,  3.540092982887960328254439790467e-10L,
4.020318667701700464612998296302e-10L,  4.544042324059585739827798668654e-10L,
3.645299460952866120296998202703e-10L,  2.776662293911361485235212513020e-12L,
1.708865101734375304910370400700e-10L,  3.909810965716415233488278047493e-10L,
7.606461848875826105025137974947e-11L,  3.263814502297453347587046149712e-10L,
1.499334758629144388918183376012e-10L,  3.771581242675818925565576303133e-10L,
1.746932950084818923507049088298e-11L,  2.837781909176306820465786987027e-10L,
3.859312847318946163435901230778e-10L,  4.601335192895268187473357720101e-10L,
2.811262558622337888849804940684e-10L,  4.060360843532416964489955306249e-10L,
8.058369357752989796958168458531e-11L,  3.725546414244147566166855921414e-10L,
1.040286509953292907344053122733e-10L,  3.094968093808145773271362531155e-10L,
4.454811192340438979284756311844e-10L,  5.676678748199027602705574110388e-11L,
2.518376833121948163898128509842e-10L,  3.907837370041422778250991189943e-10L,
7.687158710333735613246114865100e-11L,  1.334418885622867537060685125566e-10L,
1.353147719826124443836432060856e-10L,  2.825131007652335581739282335732e-10L,
4.161925466840049254333079881002e-10L,  4.265713490956410156048491599630e-10L,
2.437693664320585461575989523716e-10L,  4.466519138542116247357297503086e-10L,
3.113875178143440979746983590908e-10L,  4.910822904159495654488736486097e-11L,
2.818831329324169810481585538618e-12L,  7.767009768334052125229252512543e-12L,
3.698307026936191862258804165254e-10L,
#endif
};

/*
 * mx_atanl(x, err)
 * Table look-up algorithm
 * By K.C. Ng, March 9, 1989
 *
 * Algorithm.
 *
 * The algorithm is based on atan(x)=atan(y)+atan((x-y)/(1+x*y)).
 * We use poly1(x) to approximate atan(x) for x in [0,1/8] with
 * error (relative)
 *      |(atan(x)-poly1(x))/x|<= 2^-140
 *
 * and use poly2(x) to approximate atan(x) for x in [0,1/65] with
```



```
 * error
 *      |atan(x)-poly2(x)|<= 2^-143.7
 *
 * Here poly1 and poly2 are odd polynomial with the following form:
 *              x + x^3*(a1+x^2*(a2+...))
 *
 * (0). Purge off Inf and NaN and 0
 * (1). Reduce x to positive by atan(x) = -atan(-x).
 * (2). For x <= 1/8, use
 *      (2.1) if x < 2^(-prec/2), atan(x) =  x  with inexact flag raised
 *      (2.2) Otherwise
 *              atan(x) = poly1(x)
 * (3). For x >= 8 then (prec = 78)
 *      (3.1) if x >= 2^prec,     atan(x) = atan(inf) - pio2_lo
 *      (3.2) if x >= 2^(prec/3), atan(x) = atan(inf) - 1/x
 *      (3.3) if x >  65,         atan(x) = atan(inf) - poly2(1/x)
 *      (3.4) Otherwise,          atan(x) = atan(inf) - poly1(1/x)
 *
 * (4). Now x is in (0.125, 8)
 *      Find y that match x to 4.5 bit after binary (easy).
 *      If iy is the high word of y, then
 *              single : j = (iy - 0x3e000000) >> 19
 *              double : j = (iy - 0x3fc00000) >> 16
 *              quad   : j = (iy - 0x3ffc0000) >> 12
 *
 *      Let s = (x-y)/(1+x*y). Then
 *      atan(x) = atan(y) + poly1(s)
 *              = _TBL_atan_hi[j] + (_TBL_atan_lo[j] + poly2(s) )
 *
 *      Note. |s| <= 1.5384615385e-02 = 1/65. Maxium occurs at x = 1.03125
 *
 */

/* BEGIN CSTYLED */
#endif /* ! codereview */
/*
 * p[0] - p[16] for atan(x) =
 *              x + x^3*(p1+x^2*(p2+...))
 */
static const long double pe[] = {
        1.0L,
        0.0L,
#if defined(__sparc)
        -0.333333333333333328707404064061847748 23L,
        -4.625929269271485585084410725955082406 09e-18L,
        0.199999999999999997244442438437108648 94L,
        2.775557561562891246020470107820904644 86e-18L,
        -0.142857142857142856151586585156110231 76L,
        -9.912705577007567386212317192418005594 09e-19L,
#elif defined(__x86)
        -0.333333333255723118782043457031 25L,
        -7.761021455128987630208333319278775576 6644373e-11L,
        0.199999999953433871269226074218 75L,
        4.656612873077392578124989496139093759 38538636e-11L,
        -0.142857142840512096881866455078 125L,
        -1.663076026099068777874197038584630130 35681375e-11L,
#endif
};

static const long double p[] = {        /* p[0] - p[16] */
        1.0L,
        -3.333333333333333333333333333333333333 19278775586e-0001L,
        1.999999999999999999999999999999999898 4961390937601e-0001L,
        -1.428571428571428571428571428568669703 85846301312e-0001L,
        1.111111111111111111111111107428990944 15954427738e-0001L,
        -9.090909090909090909090879727070155492 31951421806e-0002L,
```

```
618         7.69230769230769230767699003016385628597359717046e-0002L,
619         -6.66666666666666666113842763495291228025226575259e-0002L,
620         5.88235294117646915706902204947653640091126695962e-0002L,
621         -5.26315789473657016886225044679594035524579379810e-0002L,
622         4.76190476186633969331771169790375592681525481267e-0002L,
623         -4.34782608290146274616081389793141896576997370161e-0002L,
624         3.99999968161267722260103962788865225205057218988e-0002L,
625         -3.70368536844778256320786172745225703228683638328e-0002L,
626         3.44752320396524479494062858284036892703898522150e-0002L,
627         -3.20491216046653214683721787776813360591233428081e-0002L,
628         2.67632651033434456758550618122802167256870856514e-0002L,
629 };

631 /* q[0] - q[9] */
632 static const long double qe[] = {
633         1.0L,
634         0.0L,
635 #if defined(__sparc)
636         -0.33333333333333332870740406406184774823486804962158203125L,
637         -4.62592926927148558506934546547120731253186871463421763 0e-18L,
638         0.19999999999999999972244424384371086489409208297729492 1875L,
639         2.77555756156288642682605539129568136219772203591346675 60e-18L,
640 #elif defined(__x86)
641         -0.33333333325572311878204345703125L,
642         -7.76102145512898763020833330421351509278 93e-11L,
643         0.19999999995343387126922607421875L,
644         4.65661287307739257812450757669762210686 3058e-11L,
645 #endif
646 };

648 static const long double q[] = {        /* q[0] - q[9] */
649         -3.33333333333333333333333333333333333304213515094e-0001L,
650         1.99999999999999999999999999999995075766976221077e-0001L,
651         -1.42857142857142857142857142570379604317921113079e-0001L,
652         1.11111111111111111111029238619009791279782140 77e-0001L,
653         -9.09090909090909089586854075816999506863320031460e-0002L,
654         7.69230769230756334929213246003824644696974730368e-0002L,
655         -6.66666665891924339744020135089121381681335798 56e-0002L,
656         5.88235013696778007696800252045588307023299350858e-0002L,
657         -5.25754959898164576495303840687699583228444695685e-0002L,
658 };

660 static const long double two8700 = 9.14033843895506765900208849270 1e+2618L,
661         twom8700 = 1.09405139282164366805143659376 0e-2619L, /* 2^-8700 */
662         one = 1.0L,
663         zero = 0.0L,
664         pi = 3.14159265358979323846264338327950288419716939937 51L,
665         pio2 = 1.57079632679489661923132169163975144209858469968755L,
666         pio4 = 0.78539816339744830961566084581987572104929234984377 6L,
667         pi3o4 = 2.35619449019234492884698253745962716314787704953132 9L,
461 static const long double
462 two8700  = 9.14033843895506765900208849270 1e+2618L, /* 2^8700 */
463 twom8700 = 1.09405139282164366805143659376 0e-2619L, /* 2^-8700 */
464 one   = 1.0L,
465 zero  = 0.0L,
466 pi    = 3.14159265358979323846264338327950288419716939937 51L,
467 pio2  = 1.57079632679489661923132169163975144209858469968755L,
468 pio4  = 0.78539816339744830961566084581987572104929234984377 6L,
469 pi3o4 = 2.35619449019234492884698253745962716314787704953132 9L,
668 #if defined(__sparc)
669         pi_lo = 8.67181013012378102479704402604335196876232e-35L,
670         pio2_lo = 4.33590506506189051239852201302167598438116e-35L,
671         pio4_lo = 2.16795253253094525619926100651083799219058e-35L,
672         pi3o4_lo = 6.50385759759283576859778301953251397657174e-35L;
471 pi_lo    = 8.67181013012378102479704402604335196876232e-35L,
472 pio2_lo  = 4.33590506506189051239852201302167598438116e-35L,
```



```
473 pio4_lo  = 2.16795253253094525619926100651083799219058e-35L,
474 pi3o4_lo = 6.50385759759283576859778301953251397657174e-35L;
673 #elif defined(__x86)
674         pi_lo = -5.01655761266833202355732708e-20L,
675         pio2_lo = -2.50827880633416601177866354e-20L,
676         pio4_lo = -1.25413940316708300588933177e-20L,
677         pi3o4_lo = -9.18342907192877118770525931e-20L;
476 pi_lo    = -5.01655761266833202355732708e-20L,
477 pio2_lo  = -2.50827880633416601177866354e-20L,
478 pio4_lo  = -1.25413940316708300588933177e-20L,
479 pi3o4_lo = -9.18342907192877118770525931e-20L;
678 #endif
679 /* END CSTYLED */
680 #endif /* ! codereview */

682 static long double
683 mx_atanl(long double x, long double *err)
684 {
481 mx_atanl(long double x, long double *err) {
685         long double y, z, r, s, t, w, s_h, s_l, x_h, x_l, zz[3], ee[2], z_h,
686             z_l, r_h, r_l, u, v;
687         int ix, iy, hx, i, j;
688         float fx;

690         hx = HI_XWORD(x);
691         ix = hx & (~0x80000000);

693         /* for |x| < 1/8 */
694         if (ix < 0x3ffc0000) {
695                 if (ix < 0x3ff30000) {          /* when |x| < 2**-12 */
696                         if (ix < 0x3fc60000) {  /* if |x| < 2**-prec/2 */
697                                 *err = (long double)((int)x);
494                                 *err = (long double) ((int) x);
698                                 return (x);
699                         }

701 #endif /* ! codereview */
702                         z = x * x;
703                         t = q[8];

705                         for (i = 7; i >= 0; i--)
706                                 t = q[i] + z * t;

497                         for (i = 7; i >= 0; i--) t = q[i] + z * t;
708                         t *= x * z;
709                         r = x + t;
710                         *err = t - (r - x);
711                         return (r);
712                 }

714 #endif /* ! codereview */
715                 z = x * x;

717                 /* use long double precision at p4 and on */
718                 t = p[16];

720                 for (i = 15; i >= 4; i--)
721                         t = p[i] + z * t;

503                 for (i = 15; i >= 4; i--) t = p[i] + z * t;
723                 ee[0] = z * t;

725                 x_h = x;
726                 HALF(x_h);
727                 z_h = z;
728                 HALF(z_h);
```

```
506                 x_h = x;  HALF(x_h);
507                 z_h = z;  HALF(z_h);
729                 x_l = x - x_h;
730                 z_l = (x_h * x_h - z_h);
731                 zz[0] = z;
732                 zz[1] = z_h;
733                 zz[2] = z_l + x_l * (x + x_h);

735                 /* compute (1+z*(p1+z*(p2+z*(p3+e)))) */

737                 mx_polyl(zz, pe, ee, 3);

739                 /* finally x*(1+z*(p1+...)) */
740                 r = x_h * ee[0];
741                 t = x * ee[1] + x_l * ee[0];
742                 s = t + r;
743                 *err = t - (s - r);
744                 return (s);
745         }

747 #endif /* ! codereview */
748         /* for |x| >= 8.0 */
749         if (ix >= 0x40020000) {                 /* x >=  8 */
750                 x = fabsl(x);

752 #endif /* ! codereview */
753                 if (ix >= 0x402e0000) {         /* x >=  2**47 */
754                         if (ix >= 0x408b0000)   /* x >=  2**140 */
525                         if (ix >= 0x408b0000) { /* x >=  2**140 */
755                                 y = -pio2_lo;
756                         else
527                         } else
757                                 y = one / x - pio2_lo;

759 #endif /* ! codereview */
760                         if (hx >= 0) {
761                                 t = pio2 - y;
762                                 *err = -(y - (pio2 - t));
763                         } else {
764                                 t = y - pio2;
765                                 *err = y - (pio2 + t);
766                         }

768 #endif /* ! codereview */
769                         return (t);
770                 } else {
771                         /* compute r = 1/x */
772                         r = one / x;
773                         z = r * r;
774                         x_h = x;
775                         HALF(x_h);
776                         r_h = r;
777                         HALF(r_h);
778                         z_h = z;
779                         HALF(z_h);
529                         x_h = x; HALF(x_h);
530                         r_h = r; HALF(r_h);
531                         z_h = z; HALF(z_h);
780                         r_l = r * ((x_h - x) * r_h - (x_h * r_h - one));
781                         z_l = (r_h * r_h - z_h);
782                         zz[0] = z;
783                         zz[1] = z_h;
784                         zz[2] = z_l + r_l * (r + r_h);

786 #endif /* ! codereview */
787                         if (ix < 0x40050400) {  /* 8 <  x <  65 */
```



```
788                                 /* use double precision at p4 and on */
789                                 t = p[16];

791                                 for (i = 15; i >= 4; i--)
792                                         t = p[i] + z * t;

537                                 for (i = 15; i >= 4; i--) t = p[i] + z * t;
794                                 ee[0] = z * t;
795                                 /* compute (1+z*(p1+z*(p2+z*(p3+e)))) */
796                                 mx_polyl(zz, pe, ee, 3);
797                         } else {        /* x < 65 < 2**47 */
798                                 /* use long double at q3 and on */
799                                 t = q[8];

801                                 for (i = 7; i >= 2; i--)
802                                         t = q[i] + z * t;

544                                 for (i = 7; i >= 2; i--) t = q[i] + z * t;
804                                 ee[0] = z * t;
805                                 /* compute (1+z*(q1+z*(q2+e))) */
806                                 mx_polyl(zz, qe, ee, 2);
807                         }

809 #endif /* ! codereview */
810                         /* pio2 - r*(1+...) */
811                         v = r_h * ee[0];
812                         t = pio2_lo - (r * ee[1] + r_l * ee[0]);

814 #endif /* ! codereview */
815                         if (hx >= 0) {
816                                 s = pio2 - v;
817                                 t -= (v - (pio2 - s));
818                         } else {
819                                 s = v - pio2;
820                                 t = -(t - (v - (s + pio2)));
821                         }

823 #endif /* ! codereview */
824                         w = s + t;
825                         *err = t - (w - s);
826                         return (w);
827                 }
828         }

830 #endif /* ! codereview */
831         /* now x is between 1/8 and 8 */
832         iy = (ix + 0x00000800) & 0x7ffff000;
833         j = (iy - 0x3ffc0000) >> 12;
834         ((int *)&fx)[0] = 0x3e000000 + (j << 19);
835         y = (long double)fx;
549         ((int *) &fx)[0] = 0x3e000000 + (j << 19);
550         y = (long double) fx;
836         x = fabsl(x);

838         w = (x - y);
839         v = 1.0L / (one + x * y);
840         s = w * v;
841         z = s * s;
842         /* use long double precision at q3 and on */
843         t = q[8];

845         for (i = 7; i >= 2; i--)
846                 t = q[i] + z * t;

559         for (i = 7; i >= 2; i--) t = q[i] + z * t;
848         ee[0] = z * t;
```

```
849         s_h = s;
850         HALF(s_h);
851         z_h = z;
852         HALF(z_h);
853         x_h = x;
854         HALF(x_h);
855         t = one + x * y;
856         HALF(t);
561         s_h = s; HALF(s_h);
562         z_h = z; HALF(z_h);
563         x_h = x; HALF(x_h);
564         t = one + x * y; HALF(t);
857         r = -((x_h - x) * y - (x_h * y - (t - one)));
858         s_l = -v * (s_h * r - (w - s_h * t));
859         z_l = (s_h * s_h - z_h);
860         zz[0] = z;
861         zz[1] = z_h;
862         zz[2] = z_l + s_l * (s + s_h);
863         /* compute (1+z*(q1+z*(q2+e))) by call mx_poly */
864         mx_polyl(zz, qe, ee, 2);
865         v = s_h * ee[0];
866         t = TBL_atan_lol[j] + (s * ee[1] + s_l * ee[0]);
867         u = TBL_atan_hil[j];
868         s = u + v;
869         t += (v - (s - u));
870         w = s + t;
871         *err = t - (w - s);

873 #endif /* ! codereview */
874         if (hx < 0) {
875                 w = -w;
876                 *err = -*err;
877         }

879 #endif /* ! codereview */
880         return (w);
881 }

883 long double
884 __k_atan2l(long double y, long double x, long double *w)
885 {
580 __k_atan2l(long double y, long double x, long double *w) {
886         long double t, xh, th, t1, t2, w1, w2;
887         int ix, iy, hx, hy;

889         hy = HI_XWORD(y);
890         hx = HI_XWORD(x);
891         iy = hy & ~0x80000000;
892         ix = hx & ~0x80000000;

894         *w = 0.0;

896 #endif /* ! codereview */
897         if (ix >= 0x7fff0000 || iy >= 0x7fff0000) {     /* ignore inexact */
898                 if (isnanl(x) || isnanl(y)) {
590                 if (isnanl(x) || isnanl(y))
899                         return (x * y);
900                 } else if (iy < 0x7fff0000) {
592                 else if (iy < 0x7fff0000) {
901                         if (hx >= 0) {  /* ATAN2(+-finite, +inf) is +-0 */
902                                 *w *= y;
903                                 return (*w);
904                         } else {        /* ATAN2(+-finite, -inf) is +-pi */
905                                 *w = copysignl(pi_lo, y);
906                                 return (copysignl(pi, y));
907                         }
```



```
908                 } else if (ix < 0x7fff0000) {
909                         /* ATAN2(+-inf, finite) is +-pi/2 */
910                         *w = (hy >= 0) ? pio2_lo : -pio2_lo;
911                         return ((hy >= 0) ? pio2 : -pio2);
602                         *w = (hy >= 0)? pio2_lo : -pio2_lo;
603                         return ((hy >= 0)? pio2 : -pio2);
912                 } else if (hx > 0) {    /* ATAN2(+-INF,+INF) = +-pi/4 */
913                         *w = (hy >= 0) ? pio4_lo : -pio4_lo;
914                         return ((hy >= 0) ? pio4 : -pio4);
605                         *w = (hy >= 0)? pio4_lo : -pio4_lo;
606                         return ((hy >= 0)? pio4 : -pio4);
915                 } else {                /* ATAN2(+-INF,-INF) = +-3pi/4 */
916                         *w = (hy >= 0) ? pi3o4_lo : -pi3o4_lo;
917                         return ((hy >= 0) ? pi3o4 : -pi3o4);
608                         *w = (hy >= 0)? pi3o4_lo : -pi3o4_lo;
609                         return ((hy >= 0)? pi3o4 : -pi3o4);
918                 }
919         } else if (x == zero || y == zero) {
920                 if (y == zero) {
921                         if (hx >= 0) { /* ATAN2(+-0, +(0 <= x <= inf)) is +-0 */
613                         if (hx >= 0) /* ATAN2(+-0, +(0 <= x <= inf)) is +-0 */
922                                 return (y);
923                         } else { /* ATAN2(+-0, -(0 <= x <= inf)) is +-pi */
924                                 *w = (hy >= 0) ? pi_lo : -pi_lo;
925                                 return ((hy >= 0) ? pi : -pi);
615                         else {  /* ATAN2(+-0, -(0 <= x <= inf)) is +-pi */
616                                 *w = (hy >= 0)? pi_lo : -pi_lo;
617                                 return ((hy >= 0)? pi : -pi);
926                         }
927                 } else {  /* ATAN2(+-(anything but 0 and NaN), 0) is +-pi/2 */
928                         *w = (hy >= 0) ? pio2_lo : -pio2_lo;
929                         return ((hy >= 0) ? pio2 : -pio2);
620                         *w = (hy >= 0)? pio2_lo : -pio2_lo;
621                         return ((hy >= 0)? pio2 : -pio2);
930                 }
931         } else if (iy - ix > 0x00640000) {      /* |x/y| < 2 ** -100 */
932                 *w = (hy >= 0) ? pio2_lo : -pio2_lo;
933                 return ((hy >= 0) ? pio2 : -pio2);
624                 *w = (hy >= 0)? pio2_lo : -pio2_lo;
625                 return ((hy >= 0)? pio2 : -pio2);
934         } else if (ix - iy > 0x00640000) {      /* |y/x| < 2 ** -100 */
935                 if (hx < 0) {
936                         *w = (hy >= 0) ? pi_lo : -pi_lo;
937                         return ((hy >= 0) ? pi : -pi);
628                         *w = (hy >= 0)? pi_lo : -pi_lo;
629                         return ((hy >= 0)? pi : -pi);
938                 } else {
939                         t = y / x;
940                         th = t;
941                         HALF(th);
942                         xh = x;
943                         HALF(xh);
632                         th = t; HALF(th);
633                         xh = x; HALF(xh);
944                         t1 = (x - xh) * t + xh * (t - th);
945                         t2 = y - xh * th;
946                         *w = (t2 - t1) / x;
947                         return (t);
948                 }
949         } else {
950                 if (ix >= 0x5fff3000) {
951                         x *= twom8700;
952                         y *= twom8700;
953                 } else if (ix < 0x203d0000) {
954                         x *= two8700;
955                         y *= two8700;
```

```
                }

#endif /* ! codereview */
                y = fabsl(y);
                x = fabsl(x);
                t = y / x;
                th = t;
                HALF(th);
                xh = x;
                HALF(xh);
                th = t; HALF(th);
                xh = x; HALF(xh);
                t1 = (x - xh) * t + xh * (t - th);
                t2 = y - xh * th;
                w1 = mx_atanl(t, &w2);
                w2 += (t2 - t1) / (x + y * t);

#endif /* ! codereview */
                if (hx < 0) {
                        t1 = pi - w1;
                        t2 = pi - t1;
                        w2 = (pi_lo - w2) - (w1 - t2);
                        w1 = t1;
                }

                *w = (hy >= 0) ? w2 : -w2;
                return ((hy >= 0) ? w1 : -w1);
                *w = (hy >= 0)? w2 : -w2;
                return ((hy >= 0)? w1 : -w1);
        }
}
```

_____*unchanged_portion_omitted_*

```
**********************************************************
    5577 Thu Jun  6 20:40:02 2019
new/usr/src/lib/libm/common/complex/k_cexp.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */


/* INDENT OFF */
/*
 * double __k_cexp(double x, int *n);
 * Returns the exponential of x in the form of 2**n * y, y=__k_cexp(x,&n).
 *
 * Method
 *   1. Argument reduction:
 *      Reduce x to an r so that |r| <= 0.5*ln2 ~ 0.34658.
 *      Given x, find r and integer k such that
 *
 *               x = k*ln2 + r,  |r| <= 0.5*ln2.
 *
 *      Here r will be represented as r = hi-lo for better
 *      accuracy.
 *
 *   2. Approximation of exp(r) by a special rational function on
 *      the interval [0,0.34658]:
 *      Write
 *          R(r**2) = r*(exp(r)+1)/(exp(r)-1) = 2 + r*r/6 - r**4/360 + ...
 *      We use a special Remez algorithm on [0,0.34658] to generate
 *      a polynomial of degree 5 to approximate R. The maximum error
 *      of this polynomial approximation is bounded by 2**-59. In
 *      other words,
 *          R(z) ~ 2.0 + P1*z + P2*z**2 + P3*z**3 + P4*z**4 + P5*z**5
 *      (where z=r*r, and the values of P1 to P5 are listed below)
 *      and
 *          |                    5          |     -59
 *          | 2.0+P1*z+...+P5*z   -  R(z) | <= 2
 *          |                               |
```



```
 *      The computation of exp(r) thus becomes
 *                             2*r
 *              exp(r) = 1 + -------
 *                            R - r
 *                                 r*R1(r)
 *                     = 1 + r + ----------- (for better accuracy)
 *                                2 - R1(r)
 *      where
 *                               2       4             10
 *              R1(r) = r - (P1*r  + P2*r  + ... + P5*r   ).
 *
 *   3. Return n = k and __k_cexp = exp(r).
 *
 * Special cases:
 *      exp(INF) is INF, exp(NaN) is NaN;
 *      exp(-INF) is 0, and
 *      for finite argument, only exp(0)=1 is exact.
 *
 * Range and Accuracy:
 *      When |x| is really big, say |x| > 50000, the accuracy
 *      is not important because the ultimate result will over or under
 *      flow. So we will simply replace n = 50000 and r = 0.0. For
 *      moderate size x, according to an error analysis, the error is
 *      always less than 1 ulp (unit in the last place).
 *
 * Constants:
 * The hexadecimal values are the intended ones for the following
 * constants. The decimal values may be used, provided that the
 * compiler will convert from decimal to binary accurately enough
 * to produce the hexadecimal values shown.
 */
/* INDENT ON */

#include "libm.h"                       /* __k_cexp */
#include "complex_wrapper.h"            /* HI_WORD/LO_WORD */

static const double one = 1.0,
        two128 = 3.40282366920938463463e+38;

static const double halF[2] = { 0.5, -0.5, };

static const double ln2HI[2] = {
/* INDENT OFF */
static const double
one = 1.0,
two128 = 3.40282366920938463463e+38,
halF[2] = {
        0.5, -0.5,
},
ln2HI[2] = {
        6.93147180369123816490e-01,     /* 0x3fe62e42, 0xfee00000 */
        -6.93147180369123816490e-01,    /* 0xbfe62e42, 0xfee00000 */
};

static const double ln2LO[2] = {
},
ln2LO[2] = {
        1.90821492927058770002e-10,     /* 0x3dea39ef, 0x35793c76 */
        -1.90821492927058770002e-10,    /* 0xbdea39ef, 0x35793c76 */
};

static const double
        invln2 = 1.44269504088896338700e+00,    /* 0x3ff71547, 0x652b82fe */
        P1 = 1.66666666666666019037e-01,        /* 0x3FC55555, 0x5555553E */
        P2 = -2.77777777770155933842e-03,       /* 0xBF66C16C, 0x16BEBD93 */
        P3 = 6.61375632143793436117e-05,        /* 0x3F11566A, 0xAF25DE2C */
```

```
        P4 = -1.65339022054652515390e-06,       /* 0xBEBBBD41, 0xC5D26BF1 */
        P5 = 4.13813679705723846039e-08;        /* 0x3E663769, 0x72BEA4D0 */

},
invln2 = 1.44269504088896338700e+00,    /* 0x3ff71547, 0x652b82fe */
P1 = 1.66666666666666019037e-01,        /* 0x3FC55555, 0x5555553E */
P2 = -2.77777777770155933842e-03,       /* 0xBF66C16C, 0x16BEBD93 */
P3 = 6.61375632143793436117e-05,        /* 0x3F11566A, 0xAF25DE2C */
P4 = -1.65339022054652515390e-06,       /* 0xBEBBBD41, 0xC5D26BF1 */
P5 = 4.13813679705723846039e-08;        /* 0x3E663769, 0x72BEA4D0 */
/* INDENT ON */

double
__k_cexp(double x, int *n)
{
__k_cexp(double x, int *n) {
        double hi = 0.0L, lo = 0.0L, c, t;
        int k, xsb;
        unsigned hx, lx;

        hx = HI_WORD(x);                /* high word of x */
        lx = LO_WORD(x);                /* low word of x */
        xsb = (hx >> 31) & 1;           /* sign bit of x */
        hx &= 0x7fffffff;               /* high word of |x| */

        /* filter out non-finite argument */
        if (hx >= 0x40e86a00) {         /* if |x| > 50000 */
                if (hx >= 0x7ff00000) {
                        *n = 1;

#endif /* ! codereview */
                        if (((hx & 0xfffff) | lx) != 0)
                                return (x + x);         /* NaN */
                        else
                                return ((xsb == 0) ? x : 0.0);

#endif /* ! codereview */
                        /* exp(+-inf)={inf,0} */
                }

#endif /* ! codereview */
                *n = (xsb == 0) ? 50000 : -50000;
                return (one + ln2LO[1] * ln2LO[1]);     /* generate inexact */
        }

        *n = 0;

#endif /* ! codereview */
        /* argument reduction */
        if (hx > 0x3fd62e42) {          /* if  |x| > 0.5 ln2 */
                if (hx < 0x3FF0A2B2) {  /* and |x| < 1.5 ln2 */
                        hi = x - ln2HI[xsb];
                        lo = ln2LO[xsb];
                        k = 1 - xsb - xsb;
                } else {
                        k = (int)(invln2 * x + halF[xsb]);
                        k = (int) (invln2 * x + halF[xsb]);
                        t = k;
                        hi = x - t * ln2HI[0];
                        /* t*ln2HI is exact for t<2**20 */
                        lo = t * ln2LO[0];
                }

#endif /* ! codereview */
                x = hi - lo;
                *n = k;
```



```
        } else if (hx < 0x3e300000) {   /* when |x|<2**-28 */
                return (one + x);
        } else {
        } else
                k = 0;
        }
#endif /* ! codereview */

        /* x is now in primary range */
        t = x * x;
        c = x - t * (P1 + t * (P2 + t * (P3 + t * (P4 + t * P5))));

        if (k == 0) {
        if (k == 0)
                return (one - ((x * c) / (c - 2.0) - x));
        } else {
        else {
                t = one - ((lo - (x * c) / (2.0 - c)) - hi);

#endif /* ! codereview */
                if (k > 128) {
                        t *= two128;
                        *n = k - 128;
                } else if (k > 0) {
                        HI_WORD(t) += (k << 20);
                        *n = 0;
                }

#endif /* ! codereview */
                return (t);
        }
}
```

```
**********************************************************
   10200 Thu Jun  6 20:40:02 2019
new/usr/src/lib/libm/common/complex/k_cexpl.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */


/* INDENT OFF */
/*
 * long double __k_cexpl(long double x, int *n);
 * Returns the exponential of x in the form of 2**n * y, y=__k_cexpl(x,&n).
 *
 *      1. Argument Reduction: given the input x, find r and integer k
 *         and j such that
 *                   x = (32k+j)*ln2 + r,  |r| <= (1/64)*ln2 .
 *
 *      2. expl(x) = 2^k * (2^(j/32) + 2^(j/32)*expm1(r))
 *         Note:
 *         a. expm1(r) = (2r)/(2-R), R = r - r^2*(t1 + t2*r^2)
 *         b. 2^(j/32) is represented as
 *                     exp2_32_hi[j]+exp2_32_lo[j]
 *         where
 *              exp2_32_hi[j] = 2^(j/32) rounded
 *              exp2_32_lo[j] = 2^(j/32) - exp2_32_hi[j].
 *
 * Special cases:
 *      expl(INF) is INF, expl(NaN) is NaN;
 *      expl(-INF)=  0;
 *      for finite argument, only expl(0)=1 is exact.
 *
 * Accuracy:
 *      according to an error analysis, the error is always less than
 *      an ulp (unit in the last place).
 *
 * Misc. info.
 *      When |x| is really big, say |x| > 1000000, the accuracy
```



```
 *      is not important because the ultimate result will over or under
 *      flow. So we will simply replace n = 1000000 and r = 0.0. For
 *      moderate size x, according to an error analysis, the error is
 *      always less than 1 ulp (unit in the last place).
 *
 * Constants:
 * Only decimal values are given. We assume that the compiler will convert
 * from decimal to binary accurately enough to produce the correct
 * hexadecimal values.
 */
/* INDENT ON */

#include "libm.h"                       /* __k_cexpl */
#include "complex_wrapper.h"            /* HI_XWORD */

/*
 * ln2/32 = 0.0216608493924982909192885037955680177523593791987579766912713
 */
/* INDENT OFF */
/* ln2/32 = 0.0216608493924982909192885037955680177523593791987579766912713 */
#if defined(__x86)
static const long double
        /* 43 significant bits, 21 trailing zeros */
        ln2_32hi = 2.1660849392496572818345157429575920104980468 75e-2L,
        ln2_32lo = 1.7181009433463659920976473789104487579766912713e-15L;

ln2_32hi = 2.1660849392496572818345157429575920104980468 75e-2L,
ln2_32lo = 1.7181009433463659920976473789104487579766912713e-15L;
static const long double exp2_32_hi[] = {       /* exp2_32[j] = 2^(j/32) */
        1.0000000000000000000000000e+00L, 1.0218971486541166782081522e+00L,
        1.0442737824274138402382006e+00L, 1.0671404006768236181297224e+00L,
        1.0905077326652576591003302e+00L, 1.1143867425958925362894369e+00L,
        1.1387886347566916536971221e+00L, 1.1637248587775775137938619e+00L,
        1.1892071150027210666875674e+00L, 1.2152473599804688780476325e+00L,
        1.2418578120734840485256747e+00L, 1.2690509571917332224885722e+00L,
        1.2968395546510096659215822e+00L, 1.3252366431597412945939118e+00L,
        1.3542555469368927282668852e+00L, 1.3839098819638319548151403e+00L,
        1.4142135623730950487637881e+00L, 1.4451808069770466200253470e+00L,
        1.4768261459394993113155431e+00L, 1.5091644275934227397133885e+00L,
        1.5422108254079408235859630e+00L, 1.5759808451078864864006862e+00L,
        1.6104903319492543080837174e+00L, 1.6457554781539648445110730e+00L,
        1.6817928305074290860378350e+00L, 1.7186192981224779156032914e+00L,
        1.7562521603732994831094730e+00L, 1.7947090750031071864148413e+00L,
        1.8340080864093424633989166e+00L, 1.8741676341102999013002103e+00L,
        1.9152065613971472938202589e+00L, 1.9571441241754002689657438e+00L,
        1.0000000000000000000000000e+00L,
        1.0218971486541166782081522e+00L,
        1.0442737824274138402382006e+00L,
        1.0671404006768236181297224e+00L,
        1.0905077326652576591003302e+00L,
        1.1143867425958925362894369e+00L,
        1.1387886347566916536971221e+00L,
        1.1637248587775775137938619e+00L,
        1.1892071150027210666875674e+00L,
        1.2152473599804688780476325e+00L,
        1.2418578120734840485256747e+00L,
        1.2690509571917332224885722e+00L,
        1.2968395546510096659215822e+00L,
        1.3252366431597412945939118e+00L,
        1.3542555469368927282668852e+00L,
        1.3839098819638319548151403e+00L,
        1.4142135623730950487637881e+00L,
        1.4451808069770466200253470e+00L,
        1.4768261459394993113155431e+00L,
        1.5091644275934227397133885e+00L,
        1.5422108254079408235859630e+00L,
```

```
        1.575980845107886486400686 2e+00L,
        1.610490331949254308083717 4e+00L,
        1.645755478153964844511073 0e+00L,
        1.681792830507429086037835 0e+00L,
        1.718619298122477915603291 4e+00L,
        1.756252160373299483109473 0e+00L,
        1.794709075003107186414813 e+00L,
        1.834008086409342463398916 6e+00L,
        1.874167634110299901300210 3e+00L,
        1.915206561397147293820258 9e+00L,
        1.957144124175400268965743 8e+00L,
};

#endif /* ! codereview */
static const long double exp2_32_lo[] = {
        0.000000000000000000000000000e+00L, 2.632796566718088256938252 4e-20L,
        8.376586352189519112966189 9e-20L, 3.979870577745450424920957 5e-20L,
        1.066804659665155864099304 2e-19L, 1.937600984728536044811711 4e-20L,
        6.708181945611295375127757 6e-21L, 1.971168050262918646272972 7e-20L,
        2.993258443844952368910456 9e-20L, 6.888775415303910941106191 4e-20L,
        6.800271874122537894284782 0e-20L, 6.584691737697540343974234 9e-20L,
        1.217195872751137219487600 1e-20L, 3.562525322870408711543826 0e-20L,
        3.112955155907756095630917 9e-20L, 5.751919239616477984621649 2e-20L,
        3.790065117786514159310123 9e-20L, 1.165926240569874179808011 5e-20L,
        7.136438510528469596717247 8e-20L, 5.263100371081220358878894 9e-20L,
        2.632885378873263286846058 0e-20L, 5.458395008543824278819014 1e-20L,
        9.580325437693826996071865 6e-20L, 7.683773398387424582351227 9e-21L,
        2.441596591083509382420208 7e-20L, 2.605296687101658098176972 8e-20L,
        2.687645634463255387530957 9e-21L, 1.286193015561370020170327 9e-20L,
        8.816663339403748560657229 4e-20L, 2.978861538958019094083703 7e-20L,
        5.235234161980509867742213 9e-20L, 5.257846306401046373224236 3e-20L,
        0.000000000000000000000000e+00L,
        2.632796566718088256938252 4e-20L,
        8.376586352189519112966189 9e-20L,
        3.979870577745450424920957 5e-20L,
        1.066804659665155864099304 2e-19L,
        1.937600984728536044811711 4e-20L,
        6.708181945611295375127757 6e-21L,
        1.971168050262918646272972 7e-20L,
        2.993258443844952368910456 9e-20L,
        6.888775415303910941106191 4e-20L,
        6.800271874122537894284782 0e-20L,
        6.584691737697540343974234 9e-20L,
        1.217195872751137219487600 1e-20L,
        3.562525322870408711543826 0e-20L,
        3.112955155907756095630917 9e-20L,
        5.751919239616477984621649 2e-20L,
        3.790065117786514159310123 9e-20L,
        1.165926240569874179808011 5e-20L,
        7.136438510528469596717247 8e-20L,
        5.263100371081220358878894 9e-20L,
        2.632885378873263286846058 0e-20L,
        5.458395008543824278819014 1e-20L,
        9.580325437693826996071865 6e-20L,
        7.683773398387424582351227 9e-21L,
        2.441596591083509382420208 7e-20L,
        2.605296687101658098176972 8e-20L,
        2.687645634463255387530957 9e-21L,
        1.286193015561370020170327 9e-20L,
        8.816663339403748560657229 4e-20L,
        2.978861538958019094083703 7e-20L,
        5.235234161980509867742213 9e-20L,
        5.257846306401046373224236 3e-20L,
};
#else /* sparc */
static const long double
```



```
        /* 0x3FF962E4 2FEFA39E F35793C7 00000000 */
        ln2_32hi = 2.166084939249829091928849858592451515688e-2L,
        ln2_32lo = 5.209643502595475652782654157501186731779e-27L;
ln2_32hi = 2.166084939249829091928849858592451515688e-2L,
ln2_32lo = 5.209643502595475652782654157501186731779e-27L;
static const long double exp2_32_hi[] = {       /* exp2_32[j] = 2^(j/32) */
        1.000000000000000000000000000000000000000e+0000L,
        1.021897148654116678234480134783299439782e+0000L,
        1.044273782427413840321966478739929008785e+0000L,
        1.067140400676823618169521120992809162607e+0000L,
        1.090507732665257659207010655760707978993e+0000L,
        1.114386742595892536308812956919603067800e+0000L,
        1.138788634756691653703830283841511254720e+0000L,
        1.163724858777577513813573599092185312343e+0000L,
        1.189207115002721066717499970560475915293e+0000L,
        1.215247359980468878116520251338798457624e+0000L,
        1.241857812073484048593677468726595605511e+0000L,
        1.269050957191733222554419081032338004715e+0000L,
        1.296839554651009665933754117792451159835e+0000L,
        1.325236643159741294629537095498721674113e+0000L,
        1.354255546936892728298014740140702804343e+0000L,
        1.383909881963831954872659527265192818002e+0000L,
        1.414213562373095048801688724209698078570e+0000L,
        1.445180806977046620037006241471670905678e+0000L,
        1.476826145939499311386907480374049923924e+0000L,
        1.509164427593422739766019551033193531420e+0000L,
        1.542210825407940823612291862090734841307e+0000L,
        1.575980845107886486455270160181905008906e+0000L,
        1.610490331949254308179520667357400583459e+0000L,
        1.645755478153964844518756724725822445667e+0000L,
        1.681792830507429086062250952466429790080e+0000L,
        1.718619298122477915629344376456312504516e+0000L,
        1.756252160373299483112160619375313221294e+0000L,
        1.794709075003107186427703242127781814354e+0000L,
        1.834008086409342463487083189588288856077e+0000L,
        1.874167634110299901329998949954446534439e+0000L,
        1.915206561397147293872611270295830887850e+0000L,
        1.957144124175400269018322251626871491190e+0000L,
};
_____unchanged_portion_omitted_
#endif

static const long double one = 1.0L,
static const long double
        one = 1.0L,
        two = 2.0L,
        ln2_64 = 1.083042469624914545964425189778400898568e-2L,
        invln2_32 = 4.616624130844682903551758979206054839765e+1L;

/* rational approximation coeffs for [-(ln2)/64,(ln2)/64] */
static const long double t1 = 1.666666666666666666666666666660876387437e-1L,
static const long double
        t1 =  1.666666666666666666666666666660876387437e-1L,
        t2 = -2.777777777777777777777707812093173478756e-3L,
        t3 = 6.613756613756613482074280932874221202424e-5L,
        t4 = -1.653439153392139954169609822742235851120e-6L,
        t5 = 4.175314851769539751387852116610973796053e-8L;
/* INDENT ON */

long double
__k_cexpl(long double x, int *n)
{
__k_cexpl(long double x, int *n) {
        int hx, ix, j, k;
        long double t, r;
```

```
        *n = 0;
        hx = HI_XWORD(x);
        ix = hx & 0x7fffffff;

#endif /* ! codereview */
        if (hx >= 0x7fff0000)
                return (x + x);         /* NaN of +inf */

        if (((unsigned)hx) >= 0xffff0000)
        if (((unsigned) hx) >= 0xffff0000)
                return (-one / x);      /* NaN or -inf */

#endif /* ! codereview */
        if (ix < 0x3fc30000)
                return (one + x);       /* |x|<2^-60 */

#endif /* ! codereview */
        if (hx > 0) {
                if (hx > 0x401086a0) {  /* x > 200000 */
                        *n = 200000;
                        return (one);
                }

                k = (int)(invln2_32 * (x + ln2_64));
                k = (int) (invln2_32 * (x + ln2_64));
        } else {
                if (ix > 0x401086a0) {  /* x < -200000 */
                        *n = -200000;
                        return (one);
                }

                k = (int)(invln2_32 * (x - ln2_64));
                k = (int) (invln2_32 * (x - ln2_64));
        }

#endif /* ! codereview */
        j = k & 0x1f;
        *n = k >> 5;
        t = (long double)k;
        t = (long double) k;
        x = (x - t * ln2_32hi) - t * ln2_32lo;
        t = x * x;
        r = (x - t * (t1 + t * (t2 + t * (t3 + t * (t4 + t * t5))))) - two;
        x = exp2_32_hi[j] - ((exp2_32_hi[j] * (x + x)) / r - exp2_32_lo[j]);
        k >>= 5;

#endif /* ! codereview */
        if (k > 240) {
                XFSCALE(x, 240);
                *n -= 240;
        } else if (k > 0) {
                XFSCALE(x, k);
                *n = 0;
        }

#endif /* ! codereview */
        return (x);
}
```

```
**********************************************************
   15403 Thu Jun  6 20:40:03 2019
new/usr/src/lib/libm/common/complex/k_clog_r.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

#endif /* ! codereview */
/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2005 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#include "libm.h"                       /* __k_clog_r */
#include "complex_wrapper.h"


/* INDENT OFF */
/*
 * double __k_clog_r(double x, double y, double *e);
 *
 * Compute real part of complex natural logarithm of x+iy in extra precision
 *
 * __k_clog_r returns log(hypot(x, y)) with a correction term e.
 *
 * Accuracy: 70 bits
 *
 * Method.
 * Let Z = x*x + y*y.  Z can be normalized as Z = 2^N * z,  1 <= z < 2.
 * We further break down z into 1 + zk + zh + zt, where
 *      zk = K*(2^-7) matches z to 7.5 significant bits, 0 <= K <= 2^(-7)-1
 *      zh = (z-zk) rounded to 24 bits
 *      zt = (z-zk-zh) rounded.
 *
 *          z - (1+zk)        (zh+zt)
 * Let s = ------------ = ---------------, then
 *          z + (1+zk)     2(1+zk)+zh+zt
 *                                                        z
 * log(Z) = N*log2 + log(z) = N*log2 + log(1+zk) + log(------)
 *                                                      1+zk
 *                                   1+s
 *         = N*log2 + log(1+zk) + log(---)
```



```
 *                                   1-s
 *
 *                                    1     3    1     5
 *        = N*log2 + log(1+zk) + 2s + -- (2s)  + -- (2s)  + ...
 *                                   12          80
 *
 * Note 1. For IEEE double precision,  a seven degree odd polynomial
 *              2s + P1*(2s)^3 + P2*(2s)^5 + P3*(2s)^7
 *         is generated by a special remez algorithm to
 *         approx log((1+s)/(1-s)) accurte to 72 bits.
 * Note 2. 2s can be computed accurately as s2h+s2t by
 *         r = 2/((zh+zt)+2(1+zk))
 *         s2 = r*(zh+zt)
 *         s2h = s2 rounded to float;  v = 0.5*s2h;
 *         s2t = r*((((zh-s2h*(1+zk))-v*zh)+zt)-v*zt)
 */
/* INDENT ON */

static const double zero = 0.0,
        half = 0.5,
        two = 2.0,
        two120 = 1.3292279957849158729 0e+36,            /* 2^120 */
        ln2_h = 6.93147180369123816490e-01,     /* 3fe62e42 fee00000 */
        ln2_t = 1.90821492927058770002e-10,     /* 3dea39ef 35793c76 */
        P1 = .083333333333333351554108717377986202224765262191125,
        P2 = .012499999998192275523307005746337671858964648738343 75,
        P3 = .0022321938458645656605471559987512516234702284287265625;
static const double
zero  = 0.0,
half  = 0.5,
two   = 2.0,
two120 = 1.3292279957849158729 0e+36,  /* 2^120 */
ln2_h  = 6.93147180369123816490e-01,  /* 3fe62e42 fee00000 */
ln2_t  = 1.90821492927058770002e-10,  /* 3dea39ef 35793c76 */
P1 =  .083333333333333351554108717377986202224765262191125,
P2 =  .012499999998192275523307005746337671858964648738343 75,
P3 =  .0022321938458645656605471559987512516234702284287265625;

/*
* T[2k, 2k+1] = log(1+k*2^-7) for k = 0, ..., 2^7 - 1,
* with T[2k] * 2^40 is an int
*/

/*
 * T[2k, 2k+1] = log(1+k*2^-7) for k = 0, ..., 2^7 - 1,
 * with T[2k] * 2^40 is an int
 */
#endif /* ! codereview */
static const double TBL_log1k[] = {
        0.00000000000000000000e+00, 0.00000000000000000000e+00,
        7.78214044203195953742e-03, 2.29894100462035112076e-14,
        1.55041865355087793432e-02, 4.56474807636434698847e-13,
        2.31670592811497044750e-02, 3.84673753843363762372e-13,
        3.07716586667083902285e-02, 4.52981425779092882775e-14,
        3.83188643018002039753e-02, 3.36395218465265063278e-13,
        4.58095360309016541578e-02, 3.92549008891706208826e-13,
        5.32445145181554835290e-02, 6.56799336898521766515e-13,
        6.06246218158048577607e-02, 6.29984819938331143924e-13,
        6.79506619080711971037e-02, 4.36552290856295281946e-13,
        7.52234212368421140127e-02, 7.45411685916941618656e-13,
        8.24436692109884461388e-02, 8.61451293608781447223e-14,
        8.96121586893059429713e-02, 3.81189648692113819551e-13,
        9.67296264579999842681e-02, 5.51128027471986918274e-13,
        1.03796793680885457434e-01, 7.58107392301637643358e-13,
        1.10814366339582193177e-01, 7.07921017612766061755e-13,
        1.17783035655520507134e-01, 8.62947404296943765415e-13,
```

```
        1.24703478500123310369e-01, 8.33925494898414856118e-13,
        1.31576357788617315236e-01, 1.01957352237084734958e-13,
        1.38402322858382831328e-01, 7.36304357708705134617e-13,
        1.45182009843665582594e-01, 8.32314688404647202319e-13,
        1.51916042025732167531e-01, 1.09807540998552379211e-13,
        1.58605030175749561749e-01, 8.89022343972466269900e-13,
        1.65249572894936136436e-01, 3.71026439894104998399e-13,
        1.71850256926518341061e-01, 1.40881279371111350341e-13,
        1.78407657472234859597e-01, 5.83437522462346671423e-13,
        1.84922338493379356805e-01, 6.32635858668445232946e-13,
        1.91394852999110298697e-01, 5.19155912393432989209e-13,
        1.97825743329303804785e-01, 6.16075577558872326221e-13,
        2.04215541428311553318e-01, 3.79338185766902218086e-13,
        2.10564769106895255391e-01, 4.54382278998146218219e-13,
        2.16873938300523150247e-01, 9.12093724991498410553e-14,
        2.23143551314024080057e-01, 1.85675709597960106615e-13,
        2.29374101064422575291e-01, 4.23254700234549300166e-13,
        2.35566071311950508971e-01, 8.16400106820959292914e-13,
        2.41719936886511277407e-01, 6.33890736899755317832e-13,
        2.47836163904139539227e-01, 4.41717553713155466566e-13,
        2.53915209980732470285e-01, 2.30973852175869394892e-13,
        2.59957524436686071567e-01, 2.39995404842117353465e-13,
        2.65963548496984003577e-01, 1.53937761744554075681e-13,
        2.71933715483100968413e-01, 5.40790418614551497411e-13,
        2.77868451003087102436e-01, 3.69203750820800887027e-13,
        2.83768173129828937817e-01, 8.15660529536291275782e-13,
        2.89633292582948342897e-01, 9.43339818951269030846e-14,
        2.95464212893421063200e-01, 4.14813870425856798330e-13,
        3.01261330577290209476e-01, 8.71571536970835103739e-13,
        3.07025035294827830512e-01, 8.40315630479242455758e-14,
        3.12755710003330023028e-01, 5.66865358290073900922e-13,
        3.18453731118097493891e-01, 4.37121919574291444278e-13,
        3.24119468653407238889e-01, 8.04737201185162774515e-13,
        3.29753286371669673827e-01, 7.98307987877335024112e-13,
        3.35355541920762334485e-01, 3.75495772572598557174e-13,
        3.40926586970454081893e-01, 1.39128412121975659358e-13,
        3.46466767346100823488e-01, 1.07757430375726404546e-13,
        3.51976423156884266064e-01, 2.93918591876480007730e-13,
        3.57455888921322184615e-01, 4.81589611172320539489e-13,
        3.62905493689140712377e-01, 2.27740761140395561986e-13,
        3.68325561158599157352e-01, 1.08495696229679121506e-13,
        3.73716409792905324139e-01, 6.78756682315870616582e-13,
        3.79078352934811846353e-01, 1.57612037739694350287e-13,
        3.84411698910298582632e-01, 3.34571026954408237380e-14,
        3.89716751139530970249e-01, 4.94243121138567024911e-13,
        3.94993808240542421117e-01, 3.26556988969071456956e-13,
        4.00243164126550254878e-01, 4.62452051668403792833e-13,
        4.05465108107819105498e-01, 3.45276479520397708744e-13,
        4.10659924984429380856e-01, 8.39005077851830734139e-13,
        4.15827895143593195826e-01, 1.17769787513692141889e-13,
        4.20969294643327884842e-01, 8.01751287156832458079e-13,
        4.26084395310681429692e-01, 2.18633432932159103190e-13,
        4.31173464818130014464e-01, 2.41326394913331314894e-13,
        4.36236766774527495727e-01, 3.90574622098307022265e-13,
        4.41274560804231441580e-01, 6.43787909737320689684e-13,
        4.46287102628048160113e-01, 3.71351419195920213229e-13,
        4.51274644138720759656e-01, 7.37825488412103968058e-13,
        4.56237433480964682531e-01, 6.22911850193784704748e-13,
        4.61175715121498797089e-01, 6.71369279138460114513e-13,
        4.66089729924533457961e-01, 6.57665976858006147528e-14,
        4.70979715218163619284e-01, 6.27393263311115598424e-13,
        4.75845904869856894948e-01, 1.07019317621142549209e-13,
        4.80688529345570714213e-01, 1.81193463664411114729e-13,
        4.85507815781602403149e-01, 9.84046527823262695501e-14,
        4.90303988044615834951e-01, 5.78003198945402769376e-13,
        4.95077266797125048470e-01, 7.26466128212511528295e-13,
```



```
        4.99827869555701909121e-01, 7.47420700205478712293e-13,
        5.04556010751912253909e-01, 4.83033149495532022300e-13,
        5.09261901789614057634e-01, 1.93889170049107088943e-13,
        5.13945751101346104406e-01, 8.88212395185718544720e-13,
        5.18607764207445143256e-01, 6.00488896640545761201e-13,
        5.23248143764249107335e-01, 2.98729182044413286731e-13,
        5.27867089620485785417e-01, 3.56599696633478298092e-13,
        5.32464798869114019908e-01, 3.57823965912763837621e-13,
        5.37041465896436420735e-01, 4.47233831757482468946e-13,
        5.41597282432121573947e-01, 6.22797629172251525649e-13,
        5.46132437597407260910e-01, 7.28389472720657362987e-13,
        5.50647117952394182794e-01, 2.68096466152116723636e-13,
        5.55141507539701706264e-01, 7.99886451312335479470e-13,
        5.59615787935399566777e-01, 2.31194938380053776320e-14,
        5.64070138284478161950e-01, 3.24804121719935740729e-13,
        5.68504735351780254859e-01, 8.88457219261483317716e-13,
        5.72919753561109246220e-01, 6.76262872317054154667e-13,
        5.77315365034337446559e-01, 4.86157758891509033842e-13,
        5.81691739634152327199e-01, 4.70155322075549811780e-13,
        5.86049045003164792433e-01, 4.13416470738355643357e-13,
        5.90387446602107957006e-01, 6.84176364159146659095e-14,
        5.94707107746216934174e-01, 4.75855340044306376333e-13,
        5.99008189645246602595e-01, 8.36796786747576938145e-13,
        6.03290851438032404985e-01, 5.18573553063418286042e-14,
        6.07555250243226626689e-01, 2.19132812293400917731e-13,
        6.11801541105705837253e-01, 2.87066276408616768331e-13,
        6.16029877214714360889e-01, 7.99658758518543977451e-13,
        6.20240409751204424538e-01, 6.53104313776336534177e-13,
        6.24433288011459808331e-01, 4.33692711555820529733e-13,
        6.28608659421843185555e-01, 5.30952189118357790115e-13,
        6.32766669570628437214e-01, 4.09392332186786656392e-13,
        6.36907462236194987781e-01, 8.74243839148582888557e-13,
        6.41031179420679109171e-01, 2.52181884568428814231e-13,
        6.45137961372711288277e-01, 8.73413388168702670246e-13,
        6.49227946624705509748e-01, 4.04309142530119209805e-13,
        6.53301272011958644725e-01, 7.86994033233553225797e-13,
        6.57358072708120744210e-01, 2.39285932153437645135e-13,
        6.61398482245203922503e-01, 1.61085757539324585156e-13,
        6.65422632544505177066e-01, 5.85271884362515112697e-13,
        6.69430653942072240170e-01, 5.57027128793880294600e-13,
        6.73422675211440946441e-01, 7.25773856816637653180e-13,
        6.77398823590920073912e-01, 8.86066898134949155668e-13,
        6.81359224807238206267e-01, 6.64862680714687006264e-13,
        6.85304003098281100392e-01, 6.38316151706465171657e-13,
        6.89233281238557538018e-01, 2.51442307283760746611e-13,
 0.00000000000000000000e+00,  0.00000000000000000000e+00,
 7.78214044203195953742e-03,  2.29894100462035112076e-14,
 1.55041865355087793432e-02,  4.56474807636434698847e-13,
 2.31670592811497044750e-02,  3.84673753843363762372e-13,
 3.07716586667083902285e-02,  4.52981425779092882775e-14,
 3.83188643018002039753e-02,  3.36395218465265063278e-13,
 4.58095360309016541578e-02,  3.92549008891706208826e-13,
 5.32445145181554835290e-02,  6.56799336898521766515e-13,
 6.06246218158048577607e-02,  6.29984819938331143924e-13,
 6.79506619080711971037e-02,  4.36552290856295281946e-13,
 7.52234212368421140127e-02,  7.45411685916941618656e-13,
 8.24436692109884461388e-02,  8.61451293608781447223e-14,
 8.96121586893059429713e-02,  3.81189648692113819551e-13,
 9.67296264579999842681e-02,  5.51128027471986918274e-13,
 1.03796793680885457434e-01,  7.58107392301637643358e-13,
 1.10814366339582193177e-01,  7.07921017612766061755e-13,
 1.17783035655520507134e-01,  8.62947404296943765415e-13,
 1.24703478500123310369e-01,  8.33925494898414856118e-13,
 1.31576357788617315236e-01,  1.01957352237084734958e-13,
 1.38402322858382831328e-01,  7.36304357708705134617e-13,
 1.45182009843665582594e-01,  8.32314688404647202319e-13,
```

```
1.519160420257321675 31e-01,  1.098075409985523792 11e-13,
1.586050301757495617 49e-01,  8.890223439724662699 00e-13,
1.652495728949361364 36e-01,  3.710264398941049983 99e-13,
1.718502569265183410 61e-01,  1.408812793711113503 41e-13,
1.784076574722348595 97e-01,  5.834375224623466714 23e-13,
1.849223384933793568 05e-01,  6.326358586684452329 46e-13,
1.913948529991102986 97e-01,  5.191559123934329892 09e-13,
1.978257433293038047 85e-01,  6.160755775588723262 21e-13,
2.042155414283115533 18e-01,  3.793318157669022180 86e-13,
2.105647691068952553 91e-01,  4.543822789981462182 19e-13,
2.168739383005231502 47e-01,  9.120937249914984105 53e-14,
2.231435513140240800 57e-01,  1.856757095979601066 15e-13,
2.293741010644225752 91e-01,  4.232547002345493001 66e-13,
2.355660713119505089 71e-01,  8.164001068209592929 14e-13,
2.417199368865112774 07e-01,  6.338907368997553178 32e-13,
2.478361639041395392 27e-01,  4.417175537131554665 66e-13,
2.539152099807324702 85e-01,  2.309738521758693948 92e-13,
2.599575244366860715 67e-01,  2.399954048421173534 65e-13,
2.659635484969840035 77e-01,  1.539377617445540756 81e-13,
2.719337154831009684 13e-01,  5.407904186145514974 11e-13,
2.778684510030871024 36e-01,  3.692037508208008870 27e-13,
2.837681731298289378 17e-01,  8.156605295362911275 82e-13,
2.896332925829483428 97e-01,  9.433398189512690308 46e-14,
2.954642128934210632 00e-01,  4.148131870425856798 30e-13,
3.012613305772902094 76e-01,  8.715715369708351037 39e-13,
3.070250352948278305 12e-01,  8.403156304792424557 58e-14,
3.127557100033300230 28e-01,  5.668653582900739009 22e-13,
3.184537311180974938 91e-01,  4.371219195742914442 78e-13,
3.241194686534072388 89e-01,  8.047372011851627745 15e-13,
3.297532863716696738 27e-01,  7.983079878773350241 12e-13,
3.353555419207623344 85e-01,  3.754957725725985571 74e-13,
3.409265869704540818 93e-01,  1.391284121219756593 58e-13,
3.464667673461008234 88e-01,  1.077574303757264045 46e-13,
3.519764231568842660 64e-01,  2.939185918764800077 30e-13,
3.574558889213221846 15e-01,  4.815896111723205394 89e-13,
3.629054936891407123 77e-01,  2.277407611403955619 86e-13,
3.683255611585991573 52e-01,  1.084956962296791215 06e-13,
3.737164097929053241 39e-01,  6.787566823158706165 82e-13,
3.790783529348118463 53e-01,  1.576120377396943502 87e-13,
3.844116989102985826 32e-01,  3.345710269544082373 80e-14,
3.897167511395309702 49e-01,  4.942431211385670249 11e-13,
3.949938082405424211 17e-01,  3.265569889690714569 56e-13,
4.002431641265502548 78e-01,  4.624520516684037928 33e-13,
4.054651081078191054 98e-01,  3.452764795203977087 44e-13,
4.106599249844293808 56e-01,  8.390050778518307341 39e-13,
4.158278951435931958 26e-01,  1.177697875136921418 89e-13,
4.209692946433278848 42e-01,  8.017512871568324580 79e-13,
4.260843953106814296 92e-01,  2.186334329321591031 90e-13,
4.311734648181300144 64e-01,  2.413263949133331348 94e-13,
4.362367667745274957 27e-01,  3.905746220983070222 65e-13,
4.412745608042314415 80e-01,  6.437879097373206896 84e-13,
4.462871026280481601 13e-01,  3.713514191959202132 29e-13,
4.512746441387207596 56e-01,  7.378254884121039680 58e-13,
4.562374334809646825 31e-01,  6.229118501937847047 48e-13,
4.611757151214987970 89e-01,  6.713692791384601145 13e-13,
4.660897299245334579 61e-01,  6.576659768580061475 28e-14,
4.709797152181636192 84e-01,  6.273932633111155984 24e-13,
4.758459048698568949 48e-01,  1.070193176211425492 09e-13,
4.806885293455707142 13e-01,  1.811934636644111147 29e-13,
4.855078157816024031 49e-01,  9.840465278232626955 01e-14,
4.903039880446158349 51e-01,  5.780031989454027693 76e-13,
4.950772667971250484 70e-01,  7.264661282125115282 95e-13,
4.998278695557019091 21e-01,  7.474207002054787122 93e-13,
5.045560107519122539 09e-01,  4.830331494955320223 00e-13,
5.092619017896140576 34e-01,  1.938891700491070889 43e-13,
5.139457511013461044 06e-01,  8.882123951857185447 20e-13,
```



```
5.186077642074451432 56e-01,  6.004888966405457612 01e-13,
5.232481437642491073 35e-01,  2.987291820444132867 31e-13,
5.278670896204857854 17e-01,  3.565996966334782980 92e-13,
5.324647988691140190 08e-01,  3.578239659127638376 21e-14,
5.370414658964364207 35e-01,  4.472338317574824689 46e-13,
5.415972824321215739 47e-01,  6.227976291722515256 49e-13,
5.461324375974072609 10e-01,  7.283894727206573629 87e-13,
5.506471179523941827 94e-01,  2.680964661521167236 36e-13,
5.551415075397017062 64e-01,  7.998864513123354794 70e-13,
5.596157879353995667 77e-01,  2.311949383800537763 20e-14,
5.640701382844781619 50e-01,  3.248041217199357407 29e-13,
5.685047353517802548 59e-01,  8.884572192614833177 16e-13,
5.729197535611092462 20e-01,  6.762628723170541546 67e-13,
5.773153650343374465 59e-01,  4.861577588915090338 42e-13,
5.816917396341523271 99e-01,  4.701553220755498117 80e-13,
5.860490450031647924 33e-01,  4.134164707383556433 57e-13,
5.903874466021079570 06e-01,  6.841763641591466590 95e-14,
5.947071077462169341 74e-01,  4.758553400443063763 33e-13,
5.990081896452466025 95e-01,  8.367967867475769381 45e-13,
6.032908514380324049 85e-01,  5.185735530634182860 42e-14,
6.075552502243226626 89e-01,  2.191328122934009177 31e-13,
6.118015411057058372 53e-01,  2.870662764086167683 31e-13,
6.160298772147143608 89e-01,  7.996587585185439774 51e-13,
6.202404097512044245 38e-01,  6.531043137763365341 77e-13,
6.244332880114598083 31e-01,  4.336927115558205297 33e-13,
6.286086594218431855 55e-01,  5.309521891183577901 15e-13,
6.327666695706284372 14e-01,  4.093923321867866563 92e-13,
6.369074622361949877 81e-01,  8.742438391485828885 57e-13,
6.410311794206791091 71e-01,  2.521818845684288142 31e-13,
6.451379613727112882 77e-01,  8.734133881687026702 46e-13,
6.492279466247055097 48e-01,  4.043091425301192098 05e-13,
6.533012720119586447 25e-01,  7.869940332335532257 97e-13,
6.573580727081207442 10e-01,  2.392859321534376451 35e-13,
6.613984822452039225 03e-01,  1.610857575393245851 56e-13,
6.654226325445051770 66e-01,  5.852718843625151126 97e-13,
6.694306539420722401 70e-01,  5.570271287938802946 00e-13,
6.734226752114409464 41e-01,  7.257738568166376531 80e-13,
6.773988235909200739 12e-01,  8.860668981349491556 68e-13,
6.813592248072382062 67e-01,  6.648626807146870062 64e-13,
6.853040030982811003 92e-01,  6.383161517064651716 57e-13,
6.892332812385575380 18e-01,  2.514423072837607466 11e-13,
};

/*
 * Compute N*log2 + log(1+zk+zh+zt) in extra precision
 */
static double
k_log_NKz(int N, int K, double zh, double *zt)
static double k_log_NKz(int N, int K, double zh, double *zt)
{
        double y, r, w, s2, s2h, s2t, t, zk, v, P;

        ((int *)&zk)[HIWORD] = 0x3ff00000 + (K << 13);
        ((int *)&zk)[LOWORD] = 0;
        t = zh + (*zt);
        r = two / (t + two * zk);
        s2h = s2 = r * t;
        ((int *)&s2h)[LOWORD] &= 0xe0000000;
        v = half * s2h;
        w = s2 * s2;
        s2t = r * ((((zh - s2h * zk) - v * zh) + (*zt)) - v * (*zt));
        P = s2t + (w * s2) * ((P1 + w * P2) + (w * w) * P3);
        P += N * ln2_t + TBL_log1k[K + K + 1];
        t = N * ln2_h + TBL_log1k[K + K];
        t = N*ln2_h + TBL_log1k[K+K];
        y = t + (P + s2h);
```

```
245         P -= ((y - t) - s2h);
246         *zt = P;
247         return (y);
248 }

250 double
251 __k_clog_r(double x, double y, double *er)
252 {
253         double t1, t2, t3, t4, tk, z, wh, w, zh, zk;
254         int n, k, ix, iy, iz, nx, ny, nz, i, j;
255         unsigned lx, ly;

257         ix = (((int *)&x)[HIWORD]) & ~0x80000000;
258         lx = ((unsigned *)&x)[LOWORD];
259         iy = (((int *)&y)[HIWORD]) & ~0x80000000;
260         ly = ((unsigned *)&y)[LOWORD];
261         y = fabs(y);
262         x = fabs(x);

246         y = fabs(y); x = fabs(x);
264         if (ix < iy || (ix == iy && lx < ly)) { /* force x >= y */
265                 tk = x;
266                 x = y;
267                 y = tk;
268                 n = ix, ix = iy;
269                 iy = n;
270                 n = lx, lx = ly;
271                 ly = n;
248                 tk = x;  x = y;   y = tk;
249                 n = ix, ix = iy; iy = n;
250                 n = lx, lx = ly; ly = n;
272         }

274 #endif /* ! codereview */
275         *er = zero;
276         nx = ix >> 20;
277         ny = iy >> 20;

252         nx = ix >> 20; ny = iy >> 20;
279         if (nx >= 0x7ff) {              /* x or y is Inf or NaN */
280                 if (ISINF(ix, lx))
281                         return (x);
282                 else if (ISINF(iy, ly))
283                         return (y);
284                 else
285                         return (x + y);
259                         return (x+y);
286         }

288 #endif /* ! codereview */
289 /*
290  * for tiny y (double y < 2^-35, extended y < 2^-46, quad y < 2^-70):
291  *      log(sqrt(1+y^2)) = (y^2)/2 - (y^4)/8 + ... ~= (y^2)/2
292  */
293         if ((((ix - 0x3ff00000) | lx) == 0) && ny < (0x3ff - 35)) {
294                 t2 = y * y;

296 #endif /* ! codereview */
297                 if (ny >= 565) {        /* compute er = tail of t2 */
298                         ((int *)&wh)[HIWORD] = iy;
299                         ((unsigned *)&wh)[LOWORD] = ly & 0xf8000000;
300                         *er = half * ((y - wh) * (y + wh) - (t2 - wh * wh));
301                 }

303 #endif /* ! codereview */
304                 return (half * t2);
```



```
305         }

307 #endif /* ! codereview */
308 /*
309  * x or y is subnormal or zero
310  */
311         if (nx == 0) {
312                 if ((ix | lx) == 0) {
261                 if ((ix | lx) == 0)
313                         return (-1.0 / x);
314                 } else {
263                 else {
315                         x *= two120;
316                         y *= two120;
317                         ix = ((int *)&x)[HIWORD];
318                         lx = ((unsigned *)&x)[LOWORD];
319                         iy = ((int *)&y)[HIWORD];
320                         ly = ((unsigned *)&y)[LOWORD];
321                         nx = (ix >> 20) - 120;
322                         ny = (iy >> 20) - 120;

324 #endif /* ! codereview */
325                         /* guard subnormal flush to 0 */
326                         if ((ix | lx) == 0)
327                                 return (-1.0 / x);
328                 }
329         } else if (ny == 0) {           /* y subnormal, scale it */
330                 y *= two120;
331                 iy = ((int *)&y)[HIWORD];
332                 ly = ((unsigned *)&y)[LOWORD];
333                 ny = (iy >> 20) - 120;
334         }

336 #endif /* ! codereview */
337         n = nx - ny;

339 #endif /* ! codereview */
340 /*
341  * return log(x) when y is zero or x >> y so that
342  * log(x) ~ log(sqrt(x*x+y*y)) to 27 extra bits
343  * (n > 62 for double, 78 for i386 extended, 122 for quad)
344  */
345         if (n > 62 || (iy | ly) == 0) {
346                 i = (0x000fffff & ix) | 0x3ff00000;     /* normalize x */
347                 ((int *)&x)[HIWORD] = i;
348                 i += 0x1000;
349                 ((int *)&zk)[HIWORD] = i & 0xffffe000;
350                 ((int *)&zk)[LOWORD] = 0;       /* zk matches 7.5 bits of x */
351                 z = x - zk;
352                 zh = (double)((float)z);
353                 i >>= 13;
354                 k = i & 0x7f;           /* index of zk */
355                 n = nx - 0x3ff;
356                 *er = z - zh;

358 #endif /* ! codereview */
359                 if (i >> 17) {          /* if zk = 2.0, adjust scaling */
360                         n += 1;
361                         zh *= 0.5;
362                         *er *= 0.5;
272                         zh *= 0.5;  *er *= 0.5;
363                 }

365 #endif /* ! codereview */
366                 w = k_log_NKz(n, k, zh, er);
367         } else {
```

```
368 /*
369  * compute z = x*x + y*y
370  */
371                 ix = (ix & 0xfffff) | 0x3ff00000;
372                 iy = (iy & 0xfffff) | (0x3ff00000 - (n << 20));
373                 ((int *)&x)[HIWORD] = ix;
374                 ((int *)&y)[HIWORD] = iy;
375                 t1 = x * x;
376                 t2 = y * y;
274                 ((int *)&x)[HIWORD] = ix; ((int *)&y)[HIWORD] = iy;
275                 t1 = x * x; t2 = y * y;
377                 j = ((lx >> 26) + 1) >> 1;
378                 ((int *)&wh)[HIWORD] = ix + (j >> 5);
379                 ((unsigned *)&wh)[LOWORD] = (j << 27);
380                 z = t1 + t2;

279                 z = t1+t2;
382 /*
383  * higher precision simulation x*x = t1 + t3, y*y = t2 + t4
384  */
385                 tk = wh - x;
386                 t3 = tk * tk - (two * wh * tk - (wh * wh - t1));
387                 j = ((ly >> 26) + 1) >> 1;
388                 ((int *)&wh)[HIWORD] = iy + (j >> 5);
389                 ((unsigned *)&wh)[LOWORD] = (j << 27);
390                 tk = wh - y;
391                 t4 = tk * tk - (two * wh * tk - (wh * wh - t2));

393 #endif /* ! codereview */
394 /*
395  * find zk matches z to 7.5 bits
396  */
397                 nx -= 0x3ff;
398                 iz = ((int *)&z)[HIWORD] + 0x1000;
399                 k = (iz >> 13) & 0x7f;
400                 nz = (iz >> 20) - 0x3ff;
401                 ((int *)&zk)[HIWORD] = iz & 0xffffe000;
402                 ((int *)&zk)[LOWORD] = 0;

404 #endif /* ! codereview */
405 /*
406  * order t1,t2,t3,t4 according to their size
407  */
408                 if (t2 >= fabs(t3)) {
409                         if (fabs(t3) < fabs(t4)) {
410                                 wh = t3;
411                                 t3 = t4;
412                                 t4 = wh;
290                                 wh = t3;  t3 = t4; t4 = wh;
413                         }
414                 } else {
415                         wh = t2;
416                         t2 = t3;
417                         t3 = wh;
293                         wh = t2; t2 = t3; t3 = wh;
418                 }

420 #endif /* ! codereview */
421 /*
422  * higher precision simulation: x * x + y * y = t1 + t2 + t3 + t4
423  * = zk (7 bits) + zh (24 bits) + *er (tail) and call k_log_NKz
424  */
425                 tk = t1 - zk;
426                 zh = ((tk + t2) + t3) + t4;
427                 ((int *)&zh)[LOWORD] &= 0xe0000000;
428                 w = fabs(zh);
```



```
430                 if (w >= fabs(t2)) {
295                 if (w >= fabs(t2))
431                         *er = (((tk - zh) + t2) + t3) + t4;
432                 } else {
297                 else {
433                         if (n == 0) {
434                                 wh = half * zk;
435                                 wh = (t1 - wh) - (wh - t2);
436                         } else {
301                         } else
437                                 wh = tk + t2;
438                         }

440                         if (w >= fabs(t3)) {
303                         if (w >= fabs(t3))
441                                 *er = ((wh - zh) + t3) + t4;
442                         } else {
305                         else {
443                                 z = t3;
444                                 t3 += t4;
445                                 t4 -= t3 - z;

447 #endif /* ! codereview */
448                                 if (w >= fabs(t3))
449                                         *er = ((wh - zh) + t3) + t4;
450                                 else
451                                         *er = ((wh + t3) - zh) + t4;
452                         }
453                 }

455                 if (nz == 3) {
456                         zh *= 0.125;
457                         *er *= 0.125;
458                 }

460                 if (nz == 2) {
461                         zh *= 0.25;
462                         *er *= 0.25;
463                 }

465                 if (nz == 1) {
466                         zh *= half;
467                         *er *= half;
468                 }

309                 if (nz == 3) {zh *= 0.125; *er *= 0.125; }
310                 if (nz == 2) {zh *= 0.25; *er *= 0.25; }
311                 if (nz == 1) {zh *= half; *er *= half; }
470                 nz += nx + nx;
471                 w = half * k_log_NKz(nz, k, zh, er);
472                 *er *= half;
473         }

475 #endif /* ! codereview */
476         return (w);
477 }
```

```
**********************************************************
   22888 Thu Jun  6 20:40:03 2019
new/usr/src/lib/libm/common/complex/k_clog_rl.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

#endif /* ! codereview */
/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2005 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#include "libm.h"                       /* __k_clog_rl */
#include "complex_wrapper.h"
#include "longdouble.h"


/* INDENT OFF */
/*
 * long double __k_clog_rl(long double x, long double y, long double *e);
 *
 * Compute real part of complex natural logarithm of x+iy in extra precision
 *
 * __k_clog_rl returns log(hypot(x, y)) with a correction term e.
 *
 * Accuracy: quad 140 bits, intel extended 91 bits.
 *
 * Method.
 * Assume X > Y >= 0 . Let X = 2**nx * x, Y = 2**nx * y, where 1 <= x < 2.
 * Let Z = X*X + Y*Y.  Then Z = 2**(nx+nx) * z, where z = x*x + y*y.
 * Note that z < 8.
 * Let Z = x*x + y*y.  Z can be normalized as Z = 2**N * z,  1 <= z < 2.
 * We further break down z into 1 + zk + zh + zt, where
 *      zk = K*(2**-7) matches z to 7.5 significant bits, 0 <= K <= 2**(-7)-1
 *      zh = (z-zk) rounded to half of the current significant bits
 *      zt = (z-zk-zh) rounded.
 *
 *         z - (1+zk)           (zh+zt)
 * Let s = ------------ = ---------------, then
 *         z + (1+zk)     2(1+zk)+zh+zt
 *                                                         z
```



```
 * log(Z) = N*log2 + log(z) = N*log2 + log(1+zk) + log(------)
 *                                                      1+zk
 *                                             1+s
 *        = N * log2 + log(1 +zk) + log(---)
 *                                             1-s
 *
 *                                                 3              5
 *        = N*log2 + log(1+zk) +  2s +  1/12(2s)    +   1/80(2s)  + ...
 *
 *
 * Note 1. For IEEE double precision,  a fifteen degree odd polynomial
 *              2s + P1*(2s)^3 + P2*(2s)^5 + P3*(2s)^7 + ... + P7*(2s)^15
 *        is generated by a special remez algorithm to
 *        approx log((1+s)/(1-s)) accurte to 145 bits.
 * Note 2. 2s can be computed accurately as s2h+s2t by
 *        r = 2/((zh+zt)+2(1+zk))
 *        s2 = r*(zh+zt)
 *        s2h = s2 rounded to double;  v = 0.5*s2h;
 *        s2t = r*((((zh-s2h*(1+zk))-v*zh)+zt)-v*zt)
 */
/* INDENT ON */

static const long double zero = 0.0L,
        half = 0.5L,
        two = 2.0L,
        two240 = 1.76684706477838432958329750074291851582748 39e+72L; /* 2^240 */
static const long double
zero  = 0.0L,
half  = 0.5L,
two   = 2.0L,
two240 = 1.76684706477838432958329750074291851582748 39e+72L, /* 2^240 */

/* first 48 bits of ln2 */
static const long double
        ln2_h = 0.693147180559943620892227045260369777679443359375L,
        ln2_t = 1.688525005076197806790396056774985255254120 68e-15L,
        P1 = .083333333333333333333333333333333333333341023785768375L,
        P2 = .012499999999999999999999999999996790854020757661 59375L,
        P3 = .002232142857142857142857143310092047621284490564671875L,
        P4 = .000434027777777777777774746781319264872413156956512109375L,
        P5 = .000088778409090910175633659401927718526394066546893 5546875L,
        P6 = .000018780048055589639895360927834628371268354778446533203125L,
        P7 = .000004069227854328982921366736003458838031087153635406494140625L;
ln2_h  = 0.693147180559943620892227045260369777679443359375L,
ln2_t  = 1.688525005076197806790396056774985255254120 68e-15L,
P1 = .083333333333333333333333333333333333333341023785768375L,
P2 = .012499999999999999999999999999996790854020757661 59375L,
P3 = .002232142857142857142857143310092047621284490564671875L,
P4 = .000434027777777777777774746781319264872413156956512109375L,
P5 = .000088778409090910175633659401927718526394066546893 5546875L,
P6 = .000018780048055589639895360927834628371268354778446533203125L,
P7 = .000004069227854328982921366736003458838031087153635406494140625L;

/*
 * T[2k, 2k+1] = log(1+k*2**-7) for k = 0, ..., 2**7 - 1,
 * with T[2k] * 2^48 is an int
 */

static const long double TBL_log1k[] = {
        0.000000000000000000000000000000000000000e+00L,
        0.000000000000000000000000000000000000000e+00L,
        7.782140442053275819489499554038047790527 3e-03L,
        1.673127973400507098715887598458432535122 2e-15L,
        1.550418653596352669410407543182373046875 0e-02L,
        1.727456749970610623105409118492867199031 6e-15L,
        2.316705928153339755226625129580497741699 2e-02L,
```

```
        9.80676532909666484939162416876618774748 92e-16L,
        3.07716586667510227925959043204784393310 55e-02L,
        2.66557843230327629372476064205245898136 24e-15L,
        3.83188643021341590610973071306943893432 62e-02L,
        2.44013265801799310290100270133160923323 40e-15L,
        4.58095360312924526624556165188550949096 68e-02L,
        1.75050422365109580824720426412831042631 39e-15L,
        5.32445145188091828458709642291069030761 72e-02L,
        3.10001999922955742187386340021221498911 38e-15L,
        6.06246218164336880818154895678162574768 07e-02L,
        1.15449879064247260400580939583451975128 00e-15L,
        6.79506619085046281725226435810327529907 23e-02L,
        3.12122204263419159666104391157727284173 86e-15L,
        7.52234212375846311715577030554413795471 19e-02L,
        2.89452704763692822103508975092587667431 53e-15L,
        8.24436692110737112670904025435447692871 09e-02L,
        8.80001069666124763036626986344833356768 86e-16L,
        8.96121586896860833348910091444849967956 54e-02L,
        1.04928506046023399953198953111517407992 26e-15L,
        9.67296264585506548883131472393870353698 73e-02L,
        4.57407257909248076401645167072446208706 62e-16L,
        1.03796793681642185447344672866165637969 97e-01L,
        1.37937871713089780905033660501742398220 54e-15L,
        1.10814366340289183199274702928960323333 74e-01L,
        9.30995531466394251604764733623800860369 19e-16L,
        1.17783035656380263844766886904835709887 7e-01L,
        3.19069402722256568600407971118131466908 90e-15L,
        1.24703478500954645369347417727117700805 7e-01L,
        2.59049405909765375049841104692141938900 52e-15L,
        1.31576357788716791219485457986593246459 96e-01L,
        2.48136923067070288991599179110121005672 19e-15L,
        1.38402322859118243059128872118890285491 94e-01L,
        8.92626197001482758901901215717089720003 80e-16L,
        1.45182009844496917594369733706116676330 57e-01L,
        9.79687565330034447647192010509116364800 25e-16L,
        1.51916042025838748941168887540698051452 64e-01L,
        3.22613063453735618645987494711192130181 06e-15L,
        1.58605030176637740169098833575844764709 47e-01L,
        8.43924272341049996810536219803948279987 35e-16L,
        1.65249572895305618658312596387106018066 4e-01L,
        1.54421729885289652971192259482705797461 01e-15L,
        1.71850256926656896894201054237782955169 68e-01L,
        2.32544589789181736430976570098948311327 39e-15L,
        1.78407657472817504640261176973581314086 91e-01L,
        7.92479139064537360654267769125209420368 96e-16L,
        1.84922338494011739840061636641621589660 64e-01L,
        2.52823841956017628031345146246107741260 20e-16L,
        1.91394852999628994894010247662663459777 83e-01L,
        4.59715288559898645413669207312977292692 28e-16L,
        1.97825743329918424251445685513317584991 46e-01L,
        1.45611112638568364388408380275265671915 27e-15L,
        2.04215541428688140968006337061524391174 32e-01L,
        2.75053581404913471488103942628409193377 09e-15L,
        2.10564769107346450027762330137193202972 41e-01L,
        3.18764179048259515831074812830888619289 77e-15L,
        2.16873938300611968088560388423502445220 95e-01L,
        2.39153052913732084505325802010458715994 99e-15L,
        2.23143551314208821167994756251573562622 07e-01L,
        9.34598300334058260940752530773047959962 57e-16L,
        2.29374101064845348219023435376584529876 71e-01L,
        4.81772457289669555341674255119525519746 4e-16L,
        2.35566071312764080403212574310600757598 88e-01L,
        2.82867437564463044265253808447200433817 80e-15L,
        2.41719936887143660442234249785542488098 14e-01L,
        1.50770207326612797141200524155095850529 75e-15L,
        2.47836163904580075723060872405767440795 90e-01L,
```



```
        1.18105754189334075730720301136009806231 71e-15L,
        2.53915209980963396674269461072981357574 46e-01L,
        4.74630538368336253098918349348818985607 05e-17L,
        2.59957524436924103383717010729014873504 64e-01L,
        1.96358836248381329617107167357862667959 13e-15L,
        2.65963548497136770265569793991744518280 03e-01L,
        1.17107355613254579887098879236521423335 1e-15L,
        2.71933715483640980892232619225978851318 36e-01L,
        7.77939436875307020310664215374963600043 76e-16L,
        2.77868451003453031944445683620870113372 80e-01L,
        3.27424190434930253111970923221462376921 65e-15L,
        2.83768173130642509249810245819389820098 88e-01L,
        2.08909709097653086494656192660756771124 25e-15L,
        2.89633292583040713452646741643548011779 79e-01L,
        1.96342624631388212095822407428017278236 29e-15L,
        2.95464212893833177986380178481340408325 20e-01L,
        2.69840030172757362378685644020058017506 00e-15L,
        3.01261330578160624327210825867950916290 28e-01L,
        1.15668566471236580457636706874067368038 3e-15L,
        3.07025035294909542926689027808606624603 27e-01L,
        2.31914843551272677127708573118120908018 33e-15L,
        3.12755710003894904502885765396058559417 72e-01L,
        1.98388336079429226047274206188822203988 52e-15L,
        3.18453731118534477673165383748710155487 06e-01L,
        1.38137081829841889440108145903981642682 27e-16L,
        3.24119468654210152180894510820508003234 86e-01L,
        1.82390977624961447934894747312538153764 04e-15L,
        3.29753286372465481690596789121627807617 19e-01L,
        2.50012382602279916200333447208097145522 30e-15L,
        3.35355541921135369420881033875048160552 98e-01L,
        2.46083629854593911803852145396203419109 62e-15L,
        3.40926586970592637726440443657338619232 18e-01L,
        5.72578648756123017589210904063737714580 03e-16L,
        3.46466767346207404898450477048754692077 64e-01L,
        1.17602001171137701825863419478223060699 51e-15L,
        3.51976423157175588585232617333531379699 71e-01L,
        2.59607021483892590754628964483693047905 06e-15L,
        3.57455888921801800961475237272679805755 62e-01L,
        1.97326453425286822466867905612600721848 39e-15L,
        3.62905493689368086052127182483673095703 12e-01L,
        3.67085697163493816750437254777399399781 60e-16L,
        3.68325561158705738762364489957690238952 64e-01L,
        1.91428586566409270858794454128216432476 28e-15L,
        3.73716409793583892451351857744157314300 54e-01L,
        1.88369664974971666192343891572766812813 43e-16L,
        3.79078352934968165754980873316526412963 87e-01L,
        1.29263587247231449344591754173850137258 01e-15L,
        3.84411698910330557055203826166689395904 54e-01L,
        1.48267958623631460147261400881459393417 29e-15L,
        3.89716751140024797450678306631743907928 47e-01L,
        4.15919785297371776959122588665653311896 98e-16L,
        3.94993808240865718062195810489356517791 75e-01L,
        3.26004419822587562525051823176253107323 65e-15L,
        4.00243164127012107655900763347744941711 43e-01L,
        5.99273424338647386228368514754695746627 03e-16L,
        4.05465108108163718725336366333067417144 78e-01L,
        6.63252676749131281719427215032837480083 72e-16L,
        4.10659924985267821284651290625333786010 74e-01L,
        5.64649654912550489001650824364557180778 85e-16L,
        4.15827895143710435377215617336332798004 15e-01L,
        5.30236113275618569507351763705872275094 42e-16L,
        4.20969294644127245419440441764891147613 53e-01L,
        2.39070942671974190482483633352570467911 53e-15L,
        4.26084395310898145226019551046192646026 61e-01L,
        1.91789852532854928397287005745923753098 85e-15L,
        4.31173464818368046280738781206309795379 64e-01L,
```

```
        3.2945784336977492852031005044499611665595e-15L,
        4.3623676677491474151793227065354585647583e-01L,
        3.3288311090524075754441878570852962903891e-15L,
        4.4127456080487448275562201160937547683716e-01L,
        7.4673387443005192574852544613692268411229e-16L,
        4.4628710262841764233598951250314712524414e-01L,
        1.8691966006681165218815050615460959199251e-15L,
        4.5127464413945617138779198285192251205444e-01L,
        2.4137569004002270899666314791611479063976e-15L,
        4.5623743348158640742440184112638235092163e-01L,
        1.1869564036970375473975162509216610120281e-15L,
        4.6117571512216670726047595962882041931152e-01L,
        3.4591075239659690349392915732654828400811e-15L,
        4.6608972992459740680715185590088367462158e-01L,
        1.8177514673916038857252366108673570603067e-15L,
        4.7097971521878889689105562865734100341797e-01L,
        2.1156558422273990182479555421331461933366e-15L,
        4.7584590486996347635795245878398418426514e-01L,
        4.3790725717252039722791012358345927696967e-16L,
        4.8068852934575190261057286988943815231323e-01L,
        5.0660455855585733988956280680891477171499e-18L,
        4.8550781578169832641833636444061994552612e-01L,
        2.4813834547127501689550526444948043590905e-15L,
        4.9030398804519137456736643798649311065674e-01L,
        2.4635829797216592537498738468934647345741e-15L,
        4.9507726679784980206022737547755241394043e-01L,
        1.7125377372093652812514167461480115600063e-15L,
        4.9982786955644797899367404170334339141846e-01L,
        1.3508276573735437007500942002018098437396e-15L,
        5.0455601075239187025545106735080480575562e-01L,
        3.4168028574643873701242268618467347998876e-15L,
        5.0926190178980590417268103919923305511475e-01L,
        2.0426313938800290907697638200502614622891e-15L,
        5.1394575110223428282552049495279788970947e-01L,
        3.3975485593321419703400672813719873194659e-17L,
        5.1860776420804555186805373523384332656860e-01L,
        0.0284923261130955371987633083003284697416e-17L,
        5.2324814376454753528378205373883247375488e-01L,
        3.0123302517119603836788558832352723470118e-16L,
        5.2786708962084105678513878956437110900879e-01L,
        1.3283287534282139298545497336570406582397e-15L,
        5.3246479886946929127589100971817970275879e-01L,
        2.5259803271374196253984855901484170419 21e-15L,
        5.3704146589688050994482182431966066360474e-01L,
        3.1446219074198341716354190061340477078626e-15L,
        5.4159728243274329884116014000028371810913e-01L,
        1.0727353821639001503808606766770295812627e-15L,
        5.4613243759813556721383065450936555862427e-01L,
        8.3168566554721843605240702438699163825794e-17L,
        5.5064711795266063631970610003918409347534e-01L,
        1.6429402420791657293666192255419538448840e-15L,
        5.5514150754050106684189813677221536636353e-01L,
        5.2587358222274368868380660194332415847228e-16L,
        5.5961578793542088305912329815328121185303e-01L,
        1.8032117652023735453816330571171114110385e-15L,
        5.6407013828480145889443519990891218185425e-01L,
        1.5071769490901812785299634348367857600711e-15L,
        5.6850473535266843327917740680277347564697e-01L,
        2.7879956135806418878792935692629147550413e-16L,
        5.7291975356178426181941176764667034149170e-01L,
        1.2472733449589795907271346997596471822345e-15L,
        5.7731536503482061561953742057085037231445e-01L,
        2.9886985746409486460291929160223207644146e-15L,
        5.8169173963462128540413687005639076232910e-01L,
        1.1971164738836689815783808674399742176950e-15L,
        5.8604904500357690722012193873524665832520e-01L,
```



```
        1.3016839974975520776911897855504474452726e-15L,
        5.9038744660217545856539800297468900680542e-01L,
        9.1607651870514890975077236127894522134392e-16L,
        5.9470710774668944509357970673590898513794e-01L,
        3.3444207638397932963480545729233567201211e-15L,
        5.9900818964608149030937056522816419601440e-01L,
        1.9090722294592334873060460706130642200729e-15L,
        6.0329085143808214297678205184638500213623e-01L,
        2.1193638031348149256035110177854940281795e-15L,
        6.0755525022453937822319858241826295852661e-01L,
        2.4172778865703728624133665395876418941354e-15L,
        6.1180154110599005434778518974781036376953e-01L,
        2.8491821045766810044199163148675291775782e-15L,
        6.1602987721551372146677749697118997573853e-01L,
        2.9818078843122551067455400545109858745295e-16L,
        6.2024040975185457114093878772109746932983e-01L,
        2.9577105558448461493874424529516311623184e-15L,
        6.2443328801189323939979658462107181549072e-01L,
        2.6164274215943360130441858075903119505815e-16L,
        6.2860865942237253989333112258464097976685e-01L,
        1.5978509770831895426601797458058854400463e-15L,
        6.3276666957103699928666173946112394332886e-01L,
        8.3025912472904245581515990140161946934461e-16L,
        6.3690746223706895534633076749742031097412e-01L,
        2.7627416365968377888021629180796328536455e-16L,
        6.4103117942092779912854894064366817474365e-01L,
        3.4919270523937617243719652995048419893186e-15L,
        6.4513796137358170312836591619998216629028e-01L,
        2.9985368625799347497396478978681548584217e-15L,
        6.4922794662510696639401430729776620864868e-01L,
        2.8524968256626075449136225882322854909611e-15L,
        6.5330127201274379444839723873883485794067e-01L,
        1.8443102186424720390266302263929355424008e-15L,
        6.5735807270835877602621621917933225631714e-01L,
        1.2541156738040666039091970075936624723645e-15L,
        6.6139848224536379461824253667145967483521e-01L,
        1.2136419933020381912633127333149145382797e-15L,
        6.6542263254508782210905337706208229064941e-01L,
        2.6268410392329445778904988886114643307320e-15L,
        6.6943065394262646350398426875472068786621e-01L,
        2.8037949010021747828222575923191438798877e-15L,
        6.7342267521216570003161905333399772644043e-01L,
        1.0202663413354670195383104149875619372268e-15L,
        6.7739882359180469961756898555904626846313e-01L,
        1.4411921136244383020300914304078010801275e-15L,
        6.8135922480790256372529256623238325119019e-01L,
        5.0522277899335570619054540068138110661023e-16L,
        6.8530400309891703614084690343588590621948e-01L,
        2.3804032011755313470802014258958896193599e-15L,
        6.8923328123880622797514661215245723724365e-01L,
        2.7523497677256621466659891416404053623832e-15L,
  90 0.0000000000000000000000000000000000000000e+00L,
  91 0.0000000000000000000000000000000000000000e+00L,
  92 7.7821404420532758194894995540380477905273e-03L,
  93 1.6731279734005070987158875984584325351222e-15L,
  94 1.5504186535963526694104075431823730468750e-02L,
  95 1.7274567499706106231054091184928671990316e-15L,
  96 2.3167059281533397552266251295804977416992e-02L,
  97 9.8067653290966648493916241687661877474892e-16L,
  98 3.0771658666751022792595904320478439331055e-02L,
  99 2.6655784323032762937247606420524589813624e-15L,
 100 3.8318864302134159061097307130694389343262e-02L,
 101 2.4401326580179931029010027013316092332340e-15L,
 102 4.5809536031292452662455616518855094909668e-02L,
 103 1.7505042236510958082472042641283104263139e-15L,
 104 5.3244514518809182845870964229106903076172e-02L,
```

```
3.100019999229557421873863400212214989113 8e-15L,
6.062462181643368808181548956781625747680 7e-02L,
1.154498790642472604005809395834519751280 0e-15L,
6.795066190850462817252264358103275299072 3e-02L,
3.121222042634191596661043911577272841738 6e-15L,
7.522342123758463117155770305544137954711 9e-02L,
2.894527047636928221035089750925876674315 3e-15L,
8.244366921107371126709040254354476928710 9e-02L,
8.800010696661247630366269863448333567688 6e-16L,
8.961215868968608333489100914448499679565 4e-02L,
1.049285060460233999531989531115174079922 6e-15L,
9.672962645855065488831314723938703536987 3e-02L,
4.574072579092480764016451670724462087066 2e-16L,
1.037967936816421854473446728661656379699 7e-01L,
1.379378717130897809050336605017423982205 4e-15L,
1.108143663402891831992747029289603233337 4e-01L,
9.309955314663942516047647336238008603691 9e-16L,
1.177830356563802638447668869048357009887 7e-01L,
3.190694027222565686004079711813146690890 e-15L,
1.247034785009546453693474177271127800805 7e-01L,
2.590494059097653750498411046921419389005 2e-15L,
1.315763577887167912194854579865924645996 e-01L,
2.481369230670702889915991791101210056721 9e-15L,
1.384023228591182430591288721188902854919 4e-01L,
8.926261970014827589019012157170897200038 0e-16L,
1.451820098444969175943697337061166763305 7e-01L,
9.796875653300344476719201050911636480025 e-16L,
1.519160420258387489411688875406980514526 4e-01L,
3.226130634537356186459874947111921301810 6e-15L,
1.586050301766377401690988335758447647094 7e-01L,
8.439242723410499968105362198039482799873 5e-16L,
1.652495728953056186583125963807106018066 4e-01L,
1.544217298852896529711922594827057974610 1e-15L,
1.718502569265689689420105423778295516968 e-01L,
2.325445897891817364309765700989483113273 9e-15L,
1.784076574728175046402611769735813140869 1e-01L,
7.924791390645373606542677691252094203689 6e-16L,
1.849223384940117398400616366416215896606 4e-01L,
2.528238419560176280313451462461077412602 0e-16L,
1.913948529996289948940102476626634597778 3e-01L,
4.597152885598986454136692073129772926922 8e-16L,
1.978257433299184242514456855133175849914 6e-01L,
1.456111126385683643884083802752656719152 7e-15L,
2.042155414286881409680063370615243911743 2e-01L,
2.750535814049134714881039426284091933770 9e-15L,
2.105647691073464500277623301371932029724 1e-01L,
3.187641790482595158310748128308886192897 7e-15L,
2.168739383006119680885603884235024452209 5e-01L,
2.391530529137320845053258020104587159949 9e-15L,
2.231435513142088211679947562515735626220 7e-01L,
9.345983003340582609407525307730479599625 7e-16L,
2.293741010648453482190234353765845298767 1e-01L,
4.817724572896695553416742551195255197416 4e-16L,
2.355660713127640804032125743106007575988 8e-01L,
2.828674375644630442652538084472004338178 0e-15L,
2.417199368871436604422342497855424880981 4e-01L,
1.507702073266127971412005245150958505297 5e-15L,
2.478361639045800757230608724057644079590 e-01L,
1.181057541893340757307203011360098062317 1e-15L,
2.539152099809633966742694610729813575744 6e-01L,
4.746305383683362530989183493488189856070 5e-17L,
2.599575244369241033837170107290148735046 4e-01L,
1.963588362483813296171071673578626679591 3e-15L,
2.659635484971367702655697939917445182800 3e-01L,
1.171073556132545798870988792365214223335 1e-15L,
2.719337154836409808922326192259788513183 6e-01L,
```



```
7.779394368753070203106642153749636000437 6e-16L,
2.778684510034530319444456836208701133728 0e-01L,
3.274241904349302531119709232214623769216 5e-15L,
2.837681731306425092498102458193898200988 8e-01L,
2.089097090976530864946561926607567711242 5e-15L,
2.896332925830407134526467416435480117797 9e-01L,
1.963426246313882120958224074280172782362 9e-15L,
2.954642128938331779863801784813404083252 0e-01L,
2.698400301727573623786856440200580175060 0e-15L,
3.012613305781606243272108258679509162902 8e-01L,
1.156685664712365804576367068764067368038 3e-15L,
3.070250352949095429266890278086066246032 7e-01L,
2.319148435512726771277085731181209080183 3e-15L,
3.127557100038949045028857653960585594177 2e-01L,
1.983883360794292260472742061888222039885 2e-15L,
3.184537311185344776731653837487101554870 6e-01L,
1.381370818298418894401081459039816426822 7e-16L,
3.241194686542101521808945108205080032348 6e-01L,
1.823909776249614479348947473125381537640 4e-15L,
3.297532863724654816905967891216278076171 9e-01L,
2.500123826022799162003334472080971455223 0e-15L,
3.353555419211353694208810338750481605529 8e-01L,
2.460836298545939118038521453962034191096 2e-15L,
3.409265869705926377264404436573386192321 8e-01L,
5.725786487561230175892109040637371458003 e-16L,
3.464667673462074048984504770487546920776 4e-01L,
1.176020011711377018258634194782230606995 1e-15L,
3.519764231571755885852326173335313796997 1e-01L,
2.596070214838925907546289644836930479050 6e-15L,
3.574558889218018009614752372726798057556 2e-01L,
1.973264534252868224668679056126007218483 9e-15L,
3.629054936893680805212718248367309570312 e-01L,
3.670856971634938167504372547773993997816 0e-16L,
3.683255611587057387623644899576902389526 4e-01L,
1.914285865664092708587944541282164324762 8e-15L,
3.737164097935838924513518577441573143005 4e-01L,
1.883696649749716661923438915727668128134 3e-16L,
3.790783529349681657549808733165264129638 7e-01L,
1.292635872472314493445917541738501372580 1e-15L,
3.844116989103305570552038261666893959045 4e-01L,
1.482679586236314601472614008814593934172 9e-15L,
3.897167511400247974506783066317439079284 7e-01L,
4.159197852973717769591225886656533118969 8e-16L,
3.949938082408657180621958104893565177917 5e-01L,
3.260044198225875625250518231762531073236 5e-15L,
4.002431641270121076559007633477449417114 3e-01L,
5.992734243386473862283685147546957466270 3e-16L,
4.054651081081637187253363663330674171447 8e-01L,
6.632526767491312817194272150328374808037 2e-16L,
4.106599249852678212846512906253337860107 4e-01L,
5.646496549125504890016508243645571807788 5e-16L,
4.158278951437104353772156173363279800415 e-01L,
5.302361132756185695073517637058722750944 2e-16L,
4.209692946441272454194404417648911476135 3e-01L,
2.390709426719741904824836333525704679115 3e-15L,
4.260843953108981452260195510461926460266 1e-01L,
1.917898525328549283972870057459237530998 5e-15L,
4.311734648183680462807387812063097953796 4e-01L,
3.294578433697749285203100504449961166559 5e-15L,
4.362367667749147415179322706535458564758 3e-01L,
3.328831109052407575444187857085296290388 1e-15L,
4.412745608048744827556220116093754768371 6e-01L,
7.467338744300519257485254461369226841122 9e-16L,
4.462871026284176423359895125031471252441 4e-01L,
1.869196600668116521881505061546095919925 1e-15L,
4.512746441394561713877919828519225120544 4e-01L,
```

```
2.4137569004002270899666314791611479063976e-15L,
4.5623743348158640742440184112638235092163e-01L,
1.1869564036970375473975162509216610120281e-15L,
4.6117571512216670726047595962882041931152e-01L,
3.4591075239659690349392915732654828400811e-15L,
4.6608972992459740680715185590088367462158e-01L,
1.8177514673916038857252366108673570603067e-15L,
4.7097971521878889689105562865734100341797e-01L,
2.1156558422273990182479555421331461933366e-15L,
4.7584590486996347635795245878398418426514e-01L,
4.3790725712752039722791012358345927696967e-16L,
4.8068852934575190261057286988943815231323e-01L,
5.0660455855585733988956280680894771714499e-18L,
4.8550781578169832641833636444061994552612e-01L,
2.4813834547127501689550526444948043590905e-15L,
4.9030398804519137456736643798649311065674e-01L,
2.4635829797216592537498738468934647345741e-15L,
4.9507726679784980206022737547755241394043e-01L,
1.7125377372093652812514167461480115600063e-15L,
4.9982786955644797899367404170334339141846e-01L,
1.3508276573735437007500942002018098437396e-15L,
5.0455601075239187025545106735080480575562e-01L,
3.4168028574643873701242268618467347998876e-15L,
5.0926190178980590417268103919923305511475e-01L,
2.0426313938800290907697638200502614622891e-15L,
5.1394575110223428282552049495279788970947e-01L,
3.3975485593321419703400672813719873194659e-17L,
5.1860776420804555186805373523384332656860e-01L,
8.0284923261130955371987633083003284697416e-17L,
5.2324814376454753528378205373883247375488e-01L,
3.0123302517119603836788558832352723470118e-16L,
5.2786708962084105678513878956437110900879e-01L,
1.3283287534282139298545497336570406582397e-15L,
5.3246479886946929127589100971817970275879e-01L,
2.5525980327137419625398485590148417041921e-15L,
5.3704146589688050994482182431966066360474e-01L,
3.1446219074198341716354190061340477078626e-15L,
5.4159728243274329884116014000028371810913e-01L,
1.0727353821639001503808606766770295812627e-15L,
5.4613243759813556721383065450936555862427e-01L,
8.3168566554721843605240702438699163825794e-17L,
5.5064711795266063631970610003918409347534e-01L,
1.6429404207916572936661922554195384488840e-15L,
5.5514150754050106684189813677221536636353e-01L,
5.2587358222274368868380660194332415847228e-16L,
5.5961587935420883059123298153281211853 03e-01L,
1.8032117652023735453816330571171114110385e-15L,
5.6407013828480145889443519990891218185425e-01L,
1.5071769490901812785299634348367857600711e-15L,
5.6850473535266843327917740680277347564697e-01L,
2.7879956135806418878792935692629147550413e-16L,
5.7291975356178426181941176764667034149170e-01L,
1.2472733449589795907271346997596471822345e-15L,
5.7731536503482061561953742057085037231445e-01L,
2.9886985746409486460291929160223207644146e-15L,
5.8169173963462128540413687005639076232910e-01L,
1.1971164738836689815783808674399742176950e-15L,
5.8604904500357690722012193873524665832520e-01L,
1.3016839974975520776911897855504474452726e-15L,
5.9038744660217545856539800297468900680542e-01L,
9.1607651870514890975077236127894522134392e-16L,
5.9470710774668944509357970673590898513794e-01L,
3.3444207638397932963480545729233567201211e-15L,
5.9900818964608149030937056522816419601440e-01L,
1.9090722294592334873060460706130642200729e-15L,
6.0329085143808214297678205184638500213623e-01L,
```



```
2.1193638031348149256035110177854940281795e-15L,
6.0755525022453937822319858241826295852661e-01L,
2.4172778865703728624133665395876418941354e-15L,
6.1180154110599005434778518974781036376953e-01L,
2.8491821045766810044199163148675291775782e-15L,
6.1602987721551372146677749697118997573853e-01L,
2.9818078843122551067455400545109858745295e-16L,
6.2024040975185457114093878772109746932983e-01L,
2.9577105558448461493874424529516311623184e-15L,
6.2443328801189323939979658462107181549072e-01L,
2.6164274215943360130441858075903119505815e-16L,
6.2860865942237253989333112258464097976685e-01L,
1.5978509770831895426601797458058854400463e-15L,
6.3276666957103699928666173946112394332886e-01L,
8.3025912472904245581515990140161946934461e-16L,
6.3690746223706895534633076749742031097412e-01L,
2.7627416365968377888021629180796328536455e-16L,
6.4103117942092779912854894064366817474365e-01L,
3.4919270523937617243719652995048419893186e-15L,
6.4513796137358170312836591619998216629028e-01L,
2.9985368625799347497396478978681548584217e-15L,
6.4922794662510696639401430729776620864868e-01L,
2.8524968256626075449136225882322854909611e-15L,
6.5330127201274379444839723873883485794067e-01L,
1.8443102186424720390266302263929355424008e-15L,
6.5735807270835877602621621917933225631714e-01L,
1.2541156738040666039091970075936624723645e-15L,
6.6139848224536379461824253667145967483521e-01L,
1.2136419933020381912633127333149145382797e-15L,
6.6542263254508782210905337706208229064941e-01L,
2.6268410392329445778904988886114643307320e-15L,
6.6943065394262646350398426875472068786621e-01L,
2.8037949010021747828222575923191438798877e-15L,
6.7342267521216570003161905333399772644043e-01L,
1.0202663413354670195383104149875619397268e-15L,
6.7739882359180469961756898555904626846313e-01L,
1.4411921136244383020300914304078010801275e-15L,
6.8135922480790256372529256623238325119019e-01L,
5.0522277899333570619054540068138110661023e-16L,
6.8530400309891703614084690343588590621948e-01L,
2.3804032011755313470802014258958896193599e-15L,
6.8923328123880622797514661215245723724365e-01L,
2.7523497677256621466659891416404053623832e-15L,
};

/*
 * Compute N*log2 + log(1+zk+zh+zt) in extra precision
 */
static long double
k_log_NKzl(int N, int K, long double zh, long double *zt)
static long double k_log_NKzl(int N, int K, long double zh, long double *zt)
{
        long double y, r, w, s2, s2h, s2t, t, zk, v, P;
        double dzk;

#if !defined(__x86)
        unsigned lx, ly;
        int j;
#endif

        ((int *)&dzk)[HIWORD] = 0x3ff00000 + (K << 13);
        ((int *)&dzk)[LOWORD] = 0;
        t = zh + (*zt);
        zk = (long double)dzk;
        zk = (long double) dzk;
        r = two / (t + two * zk);
```

```
381         s2h = s2 = r * t;
382 /* split s2 into correctly rounded half */

384 #if defined(__x86)
385         ((unsigned *)&s2h)[0] = 0;      /* 32 bits chopped */
386 #else

387         lx = ((unsigned *)&s2h)[2];     /* 56 bits rounded */
388         j = ((lx >> 24) + 1) >> 1;
389         ((unsigned *)&s2h)[2] = (j << 25);
390         lx = ((unsigned *)&s2h)[1];
391         ly = lx + (j >> 7);
392         ((unsigned *)&s2h)[1] = ly;
393         ((unsigned *)&s2h)[0] += (ly == 0 && lx != 0);
394         ((unsigned *)&s2h)[3] = 0;
395 #endif

397         v = half * s2h;
398         w = s2 * s2;
399         s2t = r * ((((zh - s2h * zk) - v * zh) + (*zt)) - v * (*zt));
400         P = s2t + (w * s2) * ((P1 + w * P2) + (w * w) * ((P3 + w * P4) + (w *
401             w) * (P5 + w * P6 + (w * w) * P7)));
386         P = s2t + (w * s2) * ((P1 + w * P2) + (w * w) * ((P3 + w * P4)
387                 + (w * w) * (P5 + w * P6 + (w * w) * P7)));
402         P += N * ln2_t + TBL_log1k[K + K + 1];
403         t = N * ln2_h + TBL_log1k[K + K];
389         t = N*ln2_h + TBL_log1k[K+K];
404         y = t + (P + s2h);
405         P -= ((y - t) - s2h);
406         *zt = P;
407         return (y);
408 }

410 long double
411 __k_clog_rl(long double x, long double y, long double *er)
412 {
413         long double t1, t2, t3, t4, tk, z, wh, w, zh, zk;
414         int n, k, ix, iy, iz, nx, ny, nz, i;
415         double dk;

417 #if !defined(__x86)
418         int j;
419         unsigned lx, ly;
420 #endif

422         ix = HI_XWORD(x) & ~0x80000000;
423         iy = HI_XWORD(y) & ~0x80000000;
424         y = fabsl(y);
425         x = fabsl(x);

410         y = fabsl(y); x = fabsl(x);
427         if (ix < iy || (ix < 0x7fff0000 && ix == iy && x < y)) {
428                 /* force x >= y */
429                 tk = x;
430                 x = y;
431                 y = tk;
432                 n = ix, ix = iy;
433                 iy = n;
413                 tk = x;  x = y;   y = tk;
414                 n = ix, ix = iy; iy = n;
434         }

436 #endif /* ! codereview */
437         *er = zero;
438         nx = ix >> 16;
439         ny = iy >> 16;
```



```
416         nx = ix >> 16; ny = iy >> 16;
441         if (nx >= 0x7fff) {             /* x or y is Inf or NaN */
442                 if (isinfl(x))
443                         return (x);
444                 else if (isinfl(y))
445                         return (y);
446                 else
447                         return (x + y);
423                         return (x+y);
448         }

450 #endif /* ! codereview */
451 /*
452  * for tiny y:(double y < 2^-35, extended y < 2^-46, quad y < 2^-70)
453  *
454  *      log(sqrt(1 + y**2)) =  y**2 / 2 - y**4 / 8 + ...  =  y**2 / 2
455  */
456 #if defined(__x86)
457         if (x == 1.0L && ny < (0x3fff - 46)) {
458 #else
459         if (x == 1.0L && ny < (0x3fff - 70)) {
460 #endif

462                 t2 = y * y;

464 #endif /* ! codereview */
465                 if (ny >= 8305) {       /* compute er = tail of t2 */
466                         dk = (double)y;
425                         dk = (double) y;

468 #if defined(__x86)
469                         ((unsigned *)&dk)[LOWORD] &= 0xfffe0000;
470 #endif

472                         wh = (long double)dk;
431                         wh = (long double) dk;
473                         *er = half * ((y - wh) * (y + wh) - (t2 - wh * wh));
474                 }

476 #endif /* ! codereview */
477                 return (half * t2);
478         }

480 #endif /* ! codereview */
481 /*
482  * x or y is subnormal or zero
483  */
484         if (nx == 0) {
485                 if (x == 0.0L) {
434                 if (x == 0.0L)
486                         return (-1.0L / x);
487                 } else {
436                 else {
488                         x *= two240;
489                         y *= two240;
490                         ix = HI_XWORD(x);
491                         iy = HI_XWORD(y);
492                         nx = (ix >> 16) - 240;
493                         ny = (iy >> 16) - 240;

495 #endif /* ! codereview */
496                         /* guard subnormal flush to 0 */
497                         if (x == 0.0L)
498                                 return (-1.0L / x);
499                 }
```

```
500         } else if (ny == 0) {           /* y subnormal, scale it */
501                 y *= two240;
502                 iy = HI_XWORD(y);
503                 ny = (iy >> 16) - 240;
504         }

506 #endif /* ! codereview */
507         n = nx - ny;

509 #endif /* ! codereview */
510 /*
511  * When y is zero or when x >> y, i.e.,  n > 62, 78, 122 for DBLE,
512  * EXTENDED, QUAD respectively,
513  * log(x) = log(sqrt(x * x + y * y)) to 27 extra bits.
514  */

516 #if defined(__x86)
517         if (n > 78 || y == 0.0L) {
518 #else
519         if (n > 122 || y == 0.0L) {
520 #endif

522                 XFSCALE(x, (0x3fff - (ix >> 16)));
523                 i = ((ix & 0xffff) + 0x100) >> 9;       /* 7.5 bits of x */
524                 zk = 1.0L + ((long double)i) * 0.0078125L;
443                 zk = 1.0L + ((long double) i) * 0.0078125L;
525                 z = x - zk;
526                 dk = (double)z;

528 #if defined(__x86)
529                 ((unsigned *)&dk)[LOWORD] &= 0xfffe0000;
530 #endif

532                 zh = (long double)dk;
533                 k = i & 0x7f;           /* index of zk */
534                 n = nx - 0x3fff;
535                 *er = z - zh;

537 #endif /* ! codereview */
538                 if (i == 0x80) {        /* if zk = 2.0, adjust scaling */
539                         n += 1;
540                         zh *= 0.5L;
541                         *er *= 0.5L;
455                         zh *= 0.5L;  *er *= 0.5L;
542                 }

544 #endif /* ! codereview */
545                 w = k_log_NKzl(n, k, zh, er);
546         } else {
547 /*
548  * compute z = x*x + y*y
549  */
550                 XFSCALE(x, (0x3fff - (ix >> 16)));
551                 XFSCALE(y, (0x3fff - n - (iy >> 16)));
552                 ix = (ix & 0xffff) | 0x3fff0000;
553                 iy = (iy & 0xffff) | (0x3fff0000 - (n << 16));
554                 nx -= 0x3fff;
555                 t1 = x * x;
556                 t2 = y * y;
457                 t1 = x * x; t2 = y * y;
557                 wh = x;

559 /* split x into correctly rounded half */
560 #if defined(__x86)
561                 ((unsigned *)&wh)[0] = 0;       /* 32 bits chopped */
562 #else
```



```
563                 lx = ((unsigned *)&wh)[2];      /* 56 rounded */
564                 j = ((lx >> 24) + 1) >> 1;
565                 ((unsigned *)&wh)[2] = (j << 25);
566                 lx = ((unsigned *)&wh)[1];
567                 ly = lx + (j >> 7);
568                 ((unsigned *)&wh)[1] = ly;
569                 ((unsigned *)&wh)[0] += (ly == 0 && lx != 0);
570                 ((unsigned *)&wh)[3] = 0;
571 #endif

573                 z = t1 + t2;

474                 z = t1+t2;
575 /*
576  * higher precision simulation x*x = t1 + t3, y*y = t2 + t4
577  */
578                 tk = wh - x;
579                 t3 = tk * tk - (two * wh * tk - (wh * wh - t1));
580                 wh = y;

582 /* split y into correctly rounded half */
583 #if defined(__x86)
584                 ((unsigned *)&wh)[0] = 0;       /* 32 bits chopped */
585 #else
586                 ly = ((unsigned *)&wh)[2];      /* 56 bits rounded */
587                 j = ((ly >> 24) + 1) >> 1;
588                 ((unsigned *)&wh)[2] = (j << 25);
589                 lx = ((unsigned *)&wh)[1];
590                 ly = lx + (j >> 7);
591                 ((unsigned *)&wh)[1] = ly;
592                 ((unsigned *)&wh)[0] += (ly == 0 && lx != 0);
593                 ((unsigned *)&wh)[3] = 0;
594 #endif

596                 tk = wh - y;
597                 t4 = tk * tk - (two * wh * tk - (wh * wh - t2));

599 #endif /* ! codereview */
600 /*
601  * find zk matches z to 7.5 bits
602  */
603                 iz = HI_XWORD(z);
604                 k = ((iz & 0xffff) + 0x100) >> 9;       /* 7.5 bits of x */
605                 nz = (iz >> 16) - 0x3fff + (k >> 7);
606                 k &= 0x7f;
607                 zk = 1.0L + ((long double)k) * 0.0078125L;

609                 if (nz == 1)
610                         zk += zk;
611                 else if (nz == 2)
612                         zk *= 4.0L;
613                 else if (nz == 3)
614                         zk *= 8.0L;

498                 zk = 1.0L + ((long double) k) * 0.0078125L;
499                 if (nz == 1) zk += zk;
500                 else if (nz == 2) zk *= 4.0L;
501                 else if (nz == 3) zk *= 8.0L;
616 /*
617  * order t1, t2, t3, t4 according to their size
618  */
619                 if (t2 >= fabsl(t3)) {
620                         if (fabsl(t3) < fabsl(t4)) {
621                                 wh = t3;
622                                 t3 = t4;
623                                 t4 = wh;
```

```
507                                 wh = t3;  t3 = t4; t4 = wh;
624                         }
625                 } else {
626                         wh = t2;
627                         t2 = t3;
628                         t3 = wh;
510                         wh = t2; t2 = t3; t3 = wh;
629                 }

631 #endif /* ! codereview */
632 /*
633  * higher precision simulation: x * x + y * y = t1 + t2 + t3 + t4
634  * = zk(7 bits) + zh(24 bits) + *er(tail) and call k_log_NKz
635  */
636                 tk = t1 - zk;
637                 zh = ((tk + t2) + t3) + t4;

639 /* split zh into correctly rounded half */
640 #if defined(__x86)
641                 ((unsigned *)&zh)[0] = 0;
642 #else
643                 ly = ((unsigned *)&zh)[2];
644                 j = ((ly >> 24) + 1) >> 1;
645                 ((unsigned *)&zh)[2] = (j << 25);
646                 lx = ((unsigned *)&zh)[1];
647                 ly = lx + (j >> 7);
648                 ((unsigned *)&zh)[1] = ly;
649                 ((unsigned *)&zh)[0] += (ly == 0 && lx != 0);
650                 ((unsigned *)&zh)[3] = 0;
651 #endif

653                 w = fabsl(zh);

655                 if (w >= fabsl(t2)) {
512                 if (w >= fabsl(t2))
513 {
656                         *er = (((tk - zh) + t2) + t3) + t4;
657                 } else {
515 }

517                 else {

658                         if (n == 0) {
659                                 wh = half * zk;
660                                 wh = (t1 - wh) - (wh - t2);
661                         } else {
522                         } else
662                                 wh = tk + t2;
663                         }

665                         if (w >= fabsl(t3)) {
524                         if (w >= fabsl(t3))
666                                 *er = ((wh - zh) + t3) + t4;
667                         } else {
526                         else {
668                                 z = t3;
669                                 t3 += t4;
670                                 t4 -= t3 - z;

672 #endif /* ! codereview */
673                                 if (w >= fabsl(t3))
674                                         *er = ((wh - zh) + t3) + t4;
675                                 else
676                                         *er = ((wh + t3) - zh) + t4;
677                         }
678                 }
```



```
680 #endif /* ! codereview */
681                 if (nz == 3) {
682                         zh *= 0.125L;
683                         *er *= 0.125L;
530                         zh *= 0.125L; *er *= 0.125L;
684                 } else if (nz == 2) {
685                         zh *= 0.25L;
686                         *er *= 0.25L;
532                         zh *= 0.25L; *er *= 0.25L;
687                 } else if (nz == 1) {
688                         zh *= half;
689                         *er *= half;
534                         zh *= half; *er *= half;
690                 }

692 #endif /* ! codereview */
693                 nz += nx + nx;
694                 w = half * k_log_NKzl(nz, k, zh, er);
695                 *er *= half;
696         }

698 #endif /* ! codereview */
699         return (w);
700 }
```

```
**********************************************************
   21539 Thu Jun  6 20:40:04 2019
new/usr/src/lib/libm/common/m9x/__fex_hdlr.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#undef lint
#include <signal.h>
#include <siginfo.h>
#include <ucontext.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <thread.h>
#include <math.h>
#if defined(__SUNPRO_C)
#include <sunmath.h>
#endif
#include <fenv.h>
#include "fex_handler.h"
#include "fenv_inlines.h"

#if defined(__sparc) && !defined(__sparcv9)
#include <sys/procfs.h>
#endif

/*
 * 2.x signal.h doesn’t declare sigemptyset or sigismember
 * if they’re #define’d (see sys/signal.h)
 */
/* 2.x signal.h doesn’t declare sigemptyset or sigismember
   if they’re #defined (see sys/signal.h) */
extern int sigemptyset(sigset_t *);
extern int sigismember(const sigset_t *, int);

/* external globals */
```



```
void (*__mt_fex_sync)() = NULL;         /* for synchronization with libmtsk */
#pragma weak __mt_fex_sync

void (*__libm_mt_fex_sync)() = NULL;    /* new, improved version of above */
#pragma weak __libm_mt_fex_sync

/* private variables */
static fex_handler_t main_handlers;
static int handlers_initialized = 0;
static thread_key_t handlers_key;
static mutex_t handlers_key_lock = DEFAULTMUTEX;

static struct sigaction oact = { 0, SIG_DFL };
static mutex_t hdlr_lock = DEFAULTMUTEX;
static int hdlr_installed = 0;

/* private const data */
static const int te_bit[FEX_NUM_EXC] = {
        1 << fp_trap_inexact, 1 << fp_trap_division, 1 << fp_trap_underflow,
        1 << fp_trap_overflow, 1 << fp_trap_invalid, 1 << fp_trap_invalid,
        1 << fp_trap_invalid, 1 << fp_trap_invalid, 1 << fp_trap_invalid,
        1 << fp_trap_invalid, 1 << fp_trap_invalid, 1 << fp_trap_invalid
        1 << fp_trap_inexact,
        1 << fp_trap_division,
        1 << fp_trap_underflow,
        1 << fp_trap_overflow,
        1 << fp_trap_invalid,
        1 << fp_trap_invalid,
        1 << fp_trap_invalid,
        1 << fp_trap_invalid,
        1 << fp_trap_invalid,
        1 << fp_trap_invalid,
        1 << fp_trap_invalid,
        1 << fp_trap_invalid
};

/*
 *  Return the traps to be enabled given the current handling modes
 *  and flags
 */
*  Return the traps to be enabled given the current handling modes
*  and flags
*/
static int
__fex_te_needed(struct fex_handler_data *thr_handlers, unsigned long fsr)
{
        int i, ex, te;

        /* set traps for handling modes */
        te = 0;

#endif /* ! codereview */
        for (i = 0; i < FEX_NUM_EXC; i++)
                if (thr_handlers[i].__mode != FEX_NONSTOP)
                        te |= te_bit[i];

        /* add traps for retrospective diagnostics */

#endif /* ! codereview */
        if (fex_get_log()) {
                ex = (int)__fenv_get_ex(fsr);

#endif /* ! codereview */
                if (!(ex & FE_INEXACT))
                        te |= (1 << fp_trap_inexact);
```

```
#endif /* ! codereview */
                if (!(ex & FE_UNDERFLOW))
                        te |= (1 << fp_trap_underflow);

#endif /* ! codereview */
                if (!(ex & FE_OVERFLOW))
                        te |= (1 << fp_trap_overflow);

#endif /* ! codereview */
                if (!(ex & FE_DIVBYZERO))
                        te |= (1 << fp_trap_division);

#endif /* ! codereview */
                if (!(ex & FE_INVALID))
                        te |= (1 << fp_trap_invalid);
        }

        return (te);
        return te;
}

/*
 *  The following function synchronizes with libmtsk (SPARC only, for now)
 */
 *  The following function synchronizes with libmtsk (SPARC only, for now)
*/
static void
__fex_sync_with_libmtsk(int begin, int master)
{
        static fenv_t master_env;
        static int env_initialized = 0;
        static mutex_t env_lock = DEFAULTMUTEX;

        if (begin) {
                mutex_lock(&env_lock);

#endif /* ! codereview */
                if (master) {
                        (void) fegetenv(&master_env);
                        env_initialized = 1;
                } else if (env_initialized) {
                        (void) fesetenv(&master_env);
#endif /* ! codereview */
                }

                else if (env_initialized)
                        (void) fesetenv(&master_env);
                mutex_unlock(&env_lock);
        } else if (master && fex_get_log()) {
                __fex_update_te();
#endif /* ! codereview */
        }
        else if (master && fex_get_log())
                __fex_update_te();
}

/*
 *  The following function may be used for synchronization with any
 *  internal project that manages multiple threads
 */
 *  The following function may be used for synchronization with any
 *  internal project that manages multiple threads
*/
enum __libm_mt_fex_sync_actions {
        __libm_mt_fex_start_master = 0, __libm_mt_fex_start_slave,
        __libm_mt_fex_finish_master, __libm_mt_fex_finish_slave
```



```
        __libm_mt_fex_start_master = 0,
        __libm_mt_fex_start_slave,
        __libm_mt_fex_finish_master,
        __libm_mt_fex_finish_slave
};
_____unchanged_portion_omitted_

static void
__fex_sync_with_threads(enum __libm_mt_fex_sync_actions action, struct
    __libm_mt_fex_sync_data *thr_env)
__fex_sync_with_threads(enum __libm_mt_fex_sync_actions action,
        struct __libm_mt_fex_sync_data *thr_env)
{
        switch (action) {
        case __libm_mt_fex_start_master:
                mutex_lock(&thr_env->lock);
                (void) fegetenv(&thr_env->master_env);
                thr_env->initialized = 1;
                mutex_unlock(&thr_env->lock);
                break;

        case __libm_mt_fex_start_slave:
                mutex_lock(&thr_env->lock);

#endif /* ! codereview */
                if (thr_env->initialized)
                        (void) fesetenv(&thr_env->master_env);

#endif /* ! codereview */
                mutex_unlock(&thr_env->lock);
                break;

        case __libm_mt_fex_finish_master:
#if defined(__x86)
                __fex_update_te();
#else
                if (fex_get_log())
                        __fex_update_te();
#endif
                break;

        case __libm_mt_fex_finish_slave:
#if defined(__x86)
                /*
                 * clear traps, making all accrued flags visible in status
                 * word
                 */
                /* clear traps, making all accrued flags visible in status word
                {
                        unsigned long fsr;

#endif /* ! codereview */
                        __fenv_getfsr(&fsr);
                        __fenv_set_te(fsr, 0);
                        __fenv_setfsr(&fsr);
                }
#endif
                break;
        }
}

#if defined(__sparc)

/*
 *  Code for setting or clearing interval mode on US-III and above.
 *  This is embedded as data so we don't have to mark the library
```

```
 *  as a v8plusb/v9b object.  (I could have just used one entry and
 *  modified the second word to set the bits I want, but that would
 *  have required another mutex.)
 */
*  Code for setting or clearing interval mode on US-III and above.
*  This is embedded as data so we don't have to mark the library
*  as a v8plusb/v9b object.  (I could have just used one entry and
*  modified the second word to set the bits I want, but that would
*  have required another mutex.)
*/
static const unsigned int siam[][2] = {
        { 0x81c3e008, 0x81b01020 },     /* retl, siam 0 */
        { 0x81c3e008, 0x81b01024 },     /* retl, siam 4 */
        { 0x81c3e008, 0x81b01025 },     /* retl, siam 5 */
        { 0x81c3e008, 0x81b01026 },     /* retl, siam 6 */
        { 0x81c3e008, 0x81b01027 }      /* retl, siam 7 */
};

/*
 *  If a handling mode is in effect, apply it; otherwise invoke the
 *  saved handler
 */
*  If a handling mode is in effect, apply it; otherwise invoke the
*  saved handler
*/
static void
__fex_hdlr(int sig, siginfo_t *sip, ucontext_t *uap)
{
        struct fex_handler_data *thr_handlers;
        struct sigaction act;

#endif /* ! codereview */
        void (*handler)(), (*siamp)();

#endif /* ! codereview */
        int mode, i;
        enum fex_exception e;
        fex_info_t info;
        unsigned long fsr, tmpfsr, addr;
        unsigned int gsr;

        /* determine which exception occurred */
        switch (sip->si_code) {
        case FPE_FLTDIV:
                e = fex_division;
                break;
        case FPE_FLTOVF:
                e = fex_overflow;
                break;
        case FPE_FLTUND:
                e = fex_underflow;
                break;
        case FPE_FLTRES:
                e = fex_inexact;
                break;
        case FPE_FLTINV:

#endif /* ! codereview */
                if ((int)(e = __fex_get_invalid_type(sip, uap)) < 0)
                        goto not_ieee;

#endif /* ! codereview */
                break;
        default:
                /* not an IEEE exception */
                goto not_ieee;
```



```
        }

        /* get the handling mode */
        mode = FEX_NOHANDLER;
        handler = oact.sa_handler; /* for log; just looking, no need to lock */
        thr_handlers = __fex_get_thr_handlers();

#endif /* ! codereview */
        if (thr_handlers && thr_handlers[(int)e].__mode != FEX_NOHANDLER) {
                mode = thr_handlers[(int)e].__mode;
                handler = thr_handlers[(int)e].__handler;
        }

        /* make an entry in the log of retro. diag. if need be */
        i = ((int)uap->uc_mcontext.fpregs.fpu_fsr >> 5) & 0x1f;
        __fex_mklog(uap, (char *)sip->si_addr, i, e, mode, (void *)handler);

        /* handle the exception based on the mode */
        if (mode == FEX_NOHANDLER) {
        if (mode == FEX_NOHANDLER)
                goto not_ieee;
        } else if (mode == FEX_ABORT) {
        else if (mode == FEX_ABORT)
                abort();
        } else if (mode == FEX_SIGNAL) {
        else if (mode == FEX_SIGNAL) {
                handler(sig, sip, uap);
                return;
        }

        /* custom or nonstop mode; disable traps and clear flags */
        __fenv_getfsr(&fsr);
        __fenv_set_te(fsr, 0);
        __fenv_set_ex(fsr, 0);

        /*
         * if interval mode was set, clear it, then substitute the interval
         * rounding direction and clear ns mode in the fsr
         */
        /* if interval mode was set, clear it, then substitute the
           interval rounding direction and clear ns mode in the fsr */
#ifdef __sparcv9
        gsr = uap->uc_mcontext.asrs[3];
#else
        gsr = 0;

#endif /* ! codereview */
        if (uap->uc_mcontext.xrs.xrs_id == XRS_ID)
                gsr = (*(unsigned long long *)
                    ((prxregset_t *)uap->uc_mcontext.xrs.xrs_ptr)->pr_un.
                    pr_v8p.pr_filler);
                gsr = (*(unsigned long long*)((prxregset_t*)uap->uc_mcontext.
                    xrs.xrs_ptr)->pr_un.pr_v8p.pr_filler);
#endif
        gsr = (gsr >> 25) & 7;

#endif /* ! codereview */
        if (gsr & 4) {
                siamp = (void (*)())siam[0];
                siamp = (void (*)()) siam[0];
                siamp();
                tmpfsr = fsr;
                fsr = (fsr & ~0xc0400000ul) | ((gsr & 3) << 30);
        }

#endif /* ! codereview */
```

```
        __fenv_setfsr(&fsr);

        /* decode the operation */
        __fex_get_op(sip, uap, &info);

        /* if a custom mode handler is installed, invoke it */
        if (mode == FEX_CUSTOM) {
                /* if we got here from feraiseexcept, pass dummy info */
                addr = (unsigned long)sip->si_addr;

                if (addr >= (unsigned long)feraiseexcept && addr < (unsigned
                    long)fetestexcept) {
                if (addr >= (unsigned long)feraiseexcept &&
                    addr < (unsigned long)fetestexcept) {
                        info.op = fex_other;
                        info.op1.type = info.op2.type = info.res.type =
                            fex_nodata;
                }

                /*
                 * restore interval mode if it was set, and put the original
                 * rounding direction and ns mode back in the fsr
                 */
                /* restore interval mode if it was set, and put the original
                   rounding direction and ns mode back in the fsr */
                if (gsr & 4) {
                        __fenv_setfsr(&tmpfsr);
                        siamp = (void (*)())siam[1 + (gsr & 3)];
                        siamp = (void (*)()) siam[1 + (gsr & 3)];
                        siamp();
                }

                handler(1 << (int)e, &info);

                /* restore modes in case the user's handler changed them */
                if (gsr & 4) {
                        siamp = (void (*)())siam[0];
                        siamp = (void (*)()) siam[0];
                        siamp();
                }

#endif /* ! codereview */
                __fenv_setfsr(&fsr);
        }

        /* stuff the result */
        __fex_st_result(sip, uap, &info);

        /* "or" in any exception flags and update traps */
        fsr = uap->uc_mcontext.fpregs.fpu_fsr;
        fsr |= ((info.flags & 0x1f) << 5);
        i = __fex_te_needed(thr_handlers, fsr);
        __fenv_set_te(fsr, i);
        uap->uc_mcontext.fpregs.fpu_fsr = fsr;
        return;

not_ieee:
        /* revert to the saved handler (if any) */
        mutex_lock(&hdlr_lock);
        act = oact;
        mutex_unlock(&hdlr_lock);

#endif /* ! codereview */
        switch ((unsigned long)act.sa_handler) {
        case (unsigned long)SIG_DFL:
                /* simulate trap with no handler installed */
```



```
                sigaction(SIGFPE, &act, NULL);
                kill(getpid(), SIGFPE);
                break;
#if !defined(__lint)
        case (unsigned long)SIG_IGN:
                break;
#endif
        default:
                act.sa_handler(sig, sip, uap);
        }
}

#elif defined(__x86)

#if defined(__amd64)
#define test_sse_hw             1
#else
extern int _sse_hw;

#endif /* ! codereview */
#define test_sse_hw             _sse_hw
#endif

#if !defined(REG_PC)
#define REG_PC                  EIP
#endif

/*
 *  If a handling mode is in effect, apply it; otherwise invoke the
 *  saved handler
 */
*  If a handling mode is in effect, apply it; otherwise invoke the
*  saved handler
*/
static void
__fex_hdlr(int sig, siginfo_t *sip, ucontext_t *uap)
{
        struct fex_handler_data *thr_handlers;
        struct sigaction act;

#endif /* ! codereview */
        void (*handler)() = NULL, (*simd_handler[4])();

#endif /* ! codereview */
        int mode, simd_mode[4], i, len, accrued, *ap;
        unsigned int cwsw, oldcwsw, mxcsr, oldmxcsr;
        enum fex_exception e, simd_e[4];
        fex_info_t info, simd_info[4];
        unsigned long addr;
        siginfo_t osip = *sip;
        sseinst_t inst;

        /* check for an exception caused by an SSE instruction */
        if (!(uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state.status & 0x80)) {
                len = __fex_parse_sse(uap, &inst);

#endif /* ! codereview */
                if (len == 0)
                        goto not_ieee;

                /* disable all traps and clear flags */
                __fenv_getcwsw(&oldcwsw);
                cwsw = (oldcwsw & ~0x3f) | 0x003f0000;
                __fenv_setcwsw(&cwsw);
                __fenv_getmxcsr(&oldmxcsr);
                mxcsr = (oldmxcsr & ~0x3f) | 0x1f80;
```

```
                __fenv_setmxcsr(&mxcsr);

                if ((int)inst.op & SIMD) {
                        __fex_get_simd_op(uap, &inst, simd_e, simd_info);

                        thr_handlers = __fex_get_thr_handlers();
                        addr = (unsigned long)uap->uc_mcontext.gregs[REG_PC];
                        accrued =
                            uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state.
                            mxcsr;
                        accrued = uap->uc_mcontext.fpregs.fp_reg_set.
                            fpchip_state.mxcsr;

                        e = (enum fex_exception)-1;
                        mode = FEX_NONSTOP;

#endif /* ! codereview */
                        for (i = 0; i < 4; i++) {
                                if ((int)simd_e[i] < 0)
                                        continue;

                                e = simd_e[i];
                                simd_mode[i] = FEX_NOHANDLER;
                                simd_handler[i] = oact.sa_handler;

#endif /* ! codereview */
                                if (thr_handlers &&
                                    thr_handlers[(int)e].__mode !=
                                    FEX_NOHANDLER) {
                                        simd_mode[i] =
                                            thr_handlers[(int)e].__mode;
                                        simd_handler[i] =
                                            thr_handlers[(int)e].__handler;
                                }

#endif /* ! codereview */
                                accrued &= ~te_bit[(int)e];

#endif /* ! codereview */
                                switch (simd_mode[i]) {
                                case FEX_ABORT:
                                        mode = FEX_ABORT;
                                        break;
                                case FEX_SIGNAL:

#endif /* ! codereview */
                                        if (mode != FEX_ABORT)
                                                mode = FEX_SIGNAL;

#endif /* ! codereview */
                                        handler = simd_handler[i];
                                        break;
                                case FEX_NOHANDLER:

#endif /* ! codereview */
                                        if (mode != FEX_ABORT && mode !=
                                            FEX_SIGNAL)
                                                mode = FEX_NOHANDLER;

#endif /* ! codereview */
                                        break;
                                }
                        }

#endif /* ! codereview */
                        if (e == (enum fex_exception)-1) {
```



```
                                        __fenv_setcwsw(&oldcwsw);
                                        __fenv_setmxcsr(&oldmxcsr);
                                        goto not_ieee;
                                }

                                accrued |=
                                    uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state.
                                    status;
                                accrued |= uap->uc_mcontext.fpregs.fp_reg_set.
                                    fpchip_state.status;
                                ap = __fex_accrued();
                                accrued |= *ap;
                                accrued &= 0x3d;

                                for (i = 0; i < 4; i++) {
                                        if ((int)simd_e[i] < 0)
                                                continue;

                                        __fex_mklog(uap, (char *)addr, accrued,
                                            simd_e[i], simd_mode[i],
                                            (void *)simd_handler[i]);
                                }

                                if (mode == FEX_NOHANDLER) {
                                        __fenv_setcwsw(&oldcwsw);
                                        __fenv_setmxcsr(&oldmxcsr);
                                        goto not_ieee;
                                } else if (mode == FEX_ABORT) {
                                        abort();
                                } else if (mode == FEX_SIGNAL) {
                                        __fenv_setcwsw(&oldcwsw);
                                        __fenv_setmxcsr(&oldmxcsr);
                                        handler(sig, &osip, uap);
                                        return;
                                }

                                *ap = 0;

#endif /* ! codereview */
                                for (i = 0; i < 4; i++) {
                                        if ((int)simd_e[i] < 0)
                                                continue;

                                        if (simd_mode[i] == FEX_CUSTOM) {
                                                handler(1 << (int)simd_e[i],
                                                    &simd_info[i]);
                                                __fenv_setcwsw(&cwsw);
                                                __fenv_setmxcsr(&mxcsr);
                                        }
                                }

                                __fex_st_simd_result(uap, &inst, simd_e, simd_info);

#endif /* ! codereview */
                                for (i = 0; i < 4; i++) {
                                        if ((int)simd_e[i] < 0)
                                                continue;

                                        accrued |= simd_info[i].flags;
                                }

                                if ((int)inst.op & INTREG) {
                                        /* set MMX mode */
#if defined(__amd64)
                                        uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state.
                                            sw &= ~0x3800;
```

```
                                uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state.
                                    fctw = 0;
                                uap->uc_mcontext.fpregs.fp_reg_set.
                                    fpchip_state.sw &= ~0x3800;
                                uap->uc_mcontext.fpregs.fp_reg_set.
                                    fpchip_state.fctw = 0;
#else
                                uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state.
                                    state[1] &= ~0x3800;
                                uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state.
                                    state[2] = 0;
                                uap->uc_mcontext.fpregs.fp_reg_set.
                                    fpchip_state.state[1] &= ~0x3800;
                                uap->uc_mcontext.fpregs.fp_reg_set.
                                    fpchip_state.state[2] = 0;
#endif
                        }
                } else {
                        e = __fex_get_sse_op(uap, &inst, &info);

#endif /* ! codereview */
                        if ((int)e < 0) {
                                __fenv_setcwsw(&oldcwsw);
                                __fenv_setmxcsr(&oldmxcsr);
                                goto not_ieee;
                        }

                        mode = FEX_NOHANDLER;
                        handler = oact.sa_handler;
                        thr_handlers = __fex_get_thr_handlers();

#endif /* ! codereview */
                        if (thr_handlers && thr_handlers[(int)e].__mode !=
                            FEX_NOHANDLER) {
                                mode = thr_handlers[(int)e].__mode;
                                handler = thr_handlers[(int)e].__handler;
                        }

                        addr = (unsigned long)uap->uc_mcontext.gregs[REG_PC];
                        accrued =
                            uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state.
                            mxcsr & ~te_bit[(int)e];
                        accrued |=
                            uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state.
                            status;
                        accrued = uap->uc_mcontext.fpregs.fp_reg_set.
                            fpchip_state.mxcsr & ~te_bit[(int)e];
                        accrued |= uap->uc_mcontext.fpregs.fp_reg_set.
                            fpchip_state.status;
                        ap = __fex_accrued();
                        accrued |= *ap;
                        accrued &= 0x3d;
                        __fex_mklog(uap, (char *)addr, accrued, e, mode,
                            (void *)handler);

                        if (mode == FEX_NOHANDLER) {
                                __fenv_setcwsw(&oldcwsw);
                                __fenv_setmxcsr(&oldmxcsr);
                                goto not_ieee;
                        } else if (mode == FEX_ABORT) {
                                abort();
                        } else if (mode == FEX_SIGNAL) {
                                __fenv_setcwsw(&oldcwsw);
                                __fenv_setmxcsr(&oldmxcsr);
                                handler(sig, &osip, uap);
                                return;
```



```
                        } else if (mode == FEX_CUSTOM) {
                                *ap = 0;

#endif /* ! codereview */
                                if (addr >= (unsigned long)feraiseexcept &&
                                    addr < (unsigned long)fetestexcept) {
                                        info.op = fex_other;
                                        info.op1.type = info.op2.type =
                                            info.res.type = fex_nodata;
                                }

#endif /* ! codereview */
                                handler(1 << (int)e, &info);
                                __fenv_setcwsw(&cwsw);
                                __fenv_setmxcsr(&mxcsr);
                        }

                        __fex_st_sse_result(uap, &inst, e, &info);
                        accrued |= info.flags;

#if defined(__amd64)
                        /*
                         * In 64-bit mode, the 32-bit convert-to-integer
                         * instructions zero the upper 32 bits of the
                         * destination.  (We do this here and not in
                         * __fex_st_sse_result because __fex_st_sse_result
                         * can be called from __fex_st_simd_result, too.)
                         */
                        if (inst.op == cvtss2si || inst.op == cvttss2si ||
                            inst.op == cvtsd2si || inst.op == cvttsd2si)
                                inst.op1->i[1] = 0;
#endif
                }

                /* advance the pc past the SSE instruction */
                uap->uc_mcontext.gregs[REG_PC] += len;
                goto update_state;
        }

        /* determine which exception occurred */
        __fex_get_x86_exc(sip, uap);

#endif /* ! codereview */
        switch (sip->si_code) {
        case FPE_FLTDIV:
                e = fex_division;
                break;
        case FPE_FLTOVF:
                e = fex_overflow;
                break;
        case FPE_FLTUND:
                e = fex_underflow;
                break;
        case FPE_FLTRES:
                e = fex_inexact;
                break;
        case FPE_FLTINV:

#endif /* ! codereview */
                if ((int)(e = __fex_get_invalid_type(sip, uap)) < 0)
                        goto not_ieee;

#endif /* ! codereview */
                break;
        default:
                /* not an IEEE exception */
```

```
713                 goto not_ieee;
714         }

716         /* get the handling mode */
717         mode = FEX_NOHANDLER;
718         handler = oact.sa_handler; /* for log; just looking, no need to lock */
719         thr_handlers = __fex_get_thr_handlers();

721 #endif /* ! codereview */
722         if (thr_handlers && thr_handlers[(int)e].__mode != FEX_NOHANDLER) {
723                 mode = thr_handlers[(int)e].__mode;
724                 handler = thr_handlers[(int)e].__handler;
725         }

727         /* make an entry in the log of retro. diag. if need be */
728 #if defined(__amd64)
729         addr = (unsigned
730             long)uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state.rip;
291         addr = (unsigned long)uap->uc_mcontext.fpregs.fp_reg_set.
292             fpchip_state.rip;
731 #else
732         addr = (unsigned
733             long)uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state.state[3];
294         addr = (unsigned long)uap->uc_mcontext.fpregs.fp_reg_set.
295             fpchip_state.state[3];
734 #endif
735         accrued = uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state.status &
736             ~te_bit[(int)e];

738 #endif /* ! codereview */
739         if (test_sse_hw)
740                 accrued |=
741                     uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state.mxcsr;

299                 accrued |= uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state.
300                     mxcsr;
743         ap = __fex_accrued();
744         accrued |= *ap;
745         accrued &= 0x3d;
746         __fex_mklog(uap, (char *)addr, accrued, e, mode, (void *)handler);

748         /* handle the exception based on the mode */
749         if (mode == FEX_NOHANDLER) {
307         if (mode == FEX_NOHANDLER)
750                 goto not_ieee;
751         } else if (mode == FEX_ABORT) {
309         else if (mode == FEX_ABORT)
752                 abort();
753         } else if (mode == FEX_SIGNAL) {
311         else if (mode == FEX_SIGNAL) {
754                 handler(sig, &osip, uap);
755                 return;
756         }

758         /* disable all traps and clear flags */
759         __fenv_getcwsw(&cwsw);
760         cwsw = (cwsw & ~0x3f) | 0x003f0000;
761         __fenv_setcwsw(&cwsw);

763 #endif /* ! codereview */
764         if (test_sse_hw) {
765                 __fenv_getmxcsr(&mxcsr);
766                 mxcsr = (mxcsr & ~0x3f) | 0x1f80;
767                 __fenv_setmxcsr(&mxcsr);
768         }
```



```
770 #endif /* ! codereview */
771         *ap = 0;

773         /* decode the operation */
774         __fex_get_op(sip, uap, &info);

776         /* if a custom mode handler is installed, invoke it */
777         if (mode == FEX_CUSTOM) {
778                 /* if we got here from feraiseexcept, pass dummy info */
779                 if (addr >= (unsigned long)feraiseexcept && addr < (unsigned
780                     long)fetestexcept) {
320                 if (addr >= (unsigned long)feraiseexcept &&
321                     addr < (unsigned long)fetestexcept) {
781                         info.op = fex_other;
782                         info.op1.type = info.op2.type = info.res.type =
783                             fex_nodata;
784                 }

786                 handler(1 << (int)e, &info);

788                 /* restore modes in case the user's handler changed them */
789                 __fenv_setcwsw(&cwsw);

791 #endif /* ! codereview */
792                 if (test_sse_hw)
793                         __fenv_setmxcsr(&mxcsr);
794         }

796         /* stuff the result */
797         __fex_st_result(sip, uap, &info);
798         accrued |= info.flags;

800 update_state:
801         accrued &= 0x3d;
802         i = __fex_te_needed(thr_handlers, accrued);
803         *ap = accrued & i;
804 #if defined(__amd64)
805         uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state.sw &= ~0x3d;
806         uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state.sw |= (accrued & ~i);
807         uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state.cw |= 0x3d;
808         uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state.cw &= ~i;
809 #else
810         uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state.state[1] &= ~0x3d;
811         uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state.state[1] |= (accrued &
812             ~i);
331         uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state.state[1] |=
332             (accrued & ~i);
813         uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state.state[0] |= 0x3d;
814         uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state.state[0] &= ~i;
815 #endif

817 #endif /* ! codereview */
818         if (test_sse_hw) {
819                 uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state.mxcsr &= ~0x3d;
820                 uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state.mxcsr |=
821                     0x1e80 | (accrued & ~i);
822                 uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state.mxcsr &= ~(i <<
823                     7);
336                 uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state.mxcsr &=
337                     ~(i << 7);
824         }

826 #endif /* ! codereview */
827         return;

829 not_ieee:
```

```
830         /* revert to the saved handler (if any) */
831         mutex_lock(&hdlr_lock);
832         act = oact;
833         mutex_unlock(&hdlr_lock);

835 #endif /* ! codereview */
836         switch ((unsigned long)act.sa_handler) {
837         case (unsigned long)SIG_DFL:
838                 /* simulate trap with no handler installed */
839                 sigaction(SIGFPE, &act, NULL);
840                 kill(getpid(), SIGFPE);
841                 break;
842 #if !defined(__lint)
843         case (unsigned long)SIG_IGN:
844                 break;
845 #endif
846         default:
847                 act.sa_handler(sig, &osip, uap);
848         }
849 }

850 #else
851 #error Unknown architecture
852 #endif

854 /*
855  *  Return a pointer to the thread-specific handler data, and
856  *  initialize it if necessary
857  */
345 *  Return a pointer to the thread-specific handler data, and
346 *  initialize it if necessary
347 */
858 struct fex_handler_data *
859 __fex_get_thr_handlers()
860 {
861         struct fex_handler_data *ptr;
862         unsigned long fsr;
863         int i, te;

865         if (thr_main()) {
866                 if (!handlers_initialized) {
867                         /*
868                          * initialize to FEX_NOHANDLER if trap is enabled,
869                          * FEX_NONSTOP if trap is disabled
870                          */
357                         /* initialize to FEX_NOHANDLER if trap is enabled,
358                            FEX_NONSTOP if trap is disabled */
871                         __fenv_getfsr(&fsr);
872                         te = (int)__fenv_get_te(fsr);

874 #endif /* ! codereview */
875                         for (i = 0; i < FEX_NUM_EXC; i++)
876                                 main_handlers[i].__mode = ((te & te_bit[i]) ?
877                                     FEX_NOHANDLER : FEX_NONSTOP);

361                                 main_handlers[i].__mode =
362                                         ((te & te_bit[i])? FEX_NOHANDLER : FEX_N
879                         handlers_initialized = 1;
880                 }

882                 return (main_handlers);
883         } else {
365                 return main_handlers;
366         }
367         else {
884                 ptr = NULL;
```



```
885                 mutex_lock(&handlers_key_lock);

887 #endif /* ! codereview */
888                 if (thr_getspecific(handlers_key, (void **)&ptr) != 0 &&
889                     thr_keycreate(&handlers_key, free) != 0) {
890                         mutex_unlock(&handlers_key_lock);
891                         return (NULL);
370                         return NULL;
892                 }

894 #endif /* ! codereview */
895                 mutex_unlock(&handlers_key_lock);

897 #endif /* ! codereview */
898                 if (!ptr) {
899                         if ((ptr = malloc(sizeof (fex_handler_t))) == NULL)
900                                 return (NULL);

372                         if ((ptr = (struct fex_handler_data *)
373                                 malloc(sizeof(fex_handler_t))) == NULL) {
374                                 return NULL;
375                         }
902                         if (thr_setspecific(handlers_key, (void *)ptr) != 0) {
903                                 (void) free(ptr);
904                                 return (NULL);
377                                 (void)free(ptr);
378                                 return NULL;
905                         }

907                         /*
908                          * initialize to FEX_NOHANDLER if trap is enabled,
909                          * FEX_NONSTOP if trap is disabled
910                          */
380                         /* initialize to FEX_NOHANDLER if trap is enabled,
381                            FEX_NONSTOP if trap is disabled */
911                         __fenv_getfsr(&fsr);
912                         te = (int)__fenv_get_te(fsr);

914 #endif /* ! codereview */
915                         for (i = 0; i < FEX_NUM_EXC; i++)
916                                 ptr[i].__mode = ((te & te_bit[i]) ?
917                                     FEX_NOHANDLER : FEX_NONSTOP);
384                                 ptr[i].__mode = ((te & te_bit[i])? FEX_NOHANDLER
918                 }

920                 return (ptr);
386                 return ptr;
921         }
922 }

924 /*
925  *  Update the trap enable bits according to the selected modes
926  */
391 *  Update the trap enable bits according to the selected modes
392 */
927 void
928 __fex_update_te()
929 {
930         struct fex_handler_data *thr_handlers;
931         struct sigaction act, tmpact;
932         sigset_t blocked;
933         unsigned long fsr;
934         int te;

936         /* determine which traps are needed */
937         thr_handlers = __fex_get_thr_handlers();
```

```
        __fenv_getfsr(&fsr);
        te = __fex_te_needed(thr_handlers, fsr);

        /* install __fex_hdlr as necessary */
        if (!hdlr_installed && te) {
                act.sa_handler = __fex_hdlr;
                sigemptyset(&act.sa_mask);
                act.sa_flags = SA_SIGINFO;
                sigaction(SIGFPE, &act, &tmpact);

                if (tmpact.sa_handler != __fex_hdlr) {
                if (tmpact.sa_handler != __fex_hdlr)
                {
                        mutex_lock(&hdlr_lock);
                        oact = tmpact;
                        mutex_unlock(&hdlr_lock);
                }

#endif /* ! codereview */
                hdlr_installed = 1;
        }

        /* set the new trap enable bits (only if SIGFPE is not blocked) */
        if (sigprocmask(0, NULL, &blocked) == 0 && !sigismember(&blocked,
            SIGFPE)) {
        if (sigprocmask(0, NULL, &blocked) == 0 &&
                !sigismember(&blocked, SIGFPE)) {
                __fenv_set_te(fsr, te);
                __fenv_setfsr(&fsr);
        }

        /* synchronize with libmtsk */
        __mt_fex_sync = __fex_sync_with_libmtsk;

        /* synchronize with other projects */
        __libm_mt_fex_sync = __fex_sync_with_threads;
}
```

*_____unchanged_portion_omitted_*

```
**********************************************************
   37248 Thu Jun  6 20:40:04 2019
new/usr/src/lib/libm/common/m9x/__fex_i386.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <signal.h>
#include <siginfo.h>
#include <ucontext.h>
#include <thread.h>
#include <math.h>
#if defined(__SUNPRO_C)
#include <sunmath.h>
#endif
#include <fenv.h>
#include "fex_handler.h"
#include "fenv_inlines.h"

#if defined(__amd64)
#define test_sse_hw             1
#else
/*
 * The following variable lives in libc on Solaris 10, where it
 * gets set to a nonzero value at startup time on systems with SSE.
 */
extern int _sse_hw;

#endif /* ! codereview */
#define test_sse_hw             _sse_hw
#endif

static int accrued = 0;
```



```
static thread_key_t accrued_key;
static mutex_t accrued_key_lock = DEFAULTMUTEX;

int *
__fex_accrued()
{
        int *p;

        if (thr_main()) {
                return (&accrued);
        } else {
        if (thr_main())
                return &accrued;
        else {
                p = NULL;
                mutex_lock(&accrued_key_lock);

#endif /* ! codereview */
                if (thr_getspecific(accrued_key, (void **)&p) != 0 &&
                    thr_keycreate(&accrued_key, free) != 0) {
                        mutex_unlock(&accrued_key_lock);
                        return (NULL);
                        return NULL;
                }

#endif /* ! codereview */
                mutex_unlock(&accrued_key_lock);

#endif /* ! codereview */
                if (!p) {
                        if ((p = (int *)malloc(sizeof (int))) == NULL)
                                return (NULL);

                        if ((p = (int*) malloc(sizeof(int))) == NULL)
                                return NULL;
                        if (thr_setspecific(accrued_key, (void *)p) != 0) {
                                (void) free(p);
                                return (NULL);
                                (void)free(p);
                                return NULL;
                        }

#endif /* ! codereview */
                        *p = 0;
                }

                return (p);
                return p;
        }
}

void
__fenv_getfsr(unsigned long *fsr)
{
        unsigned int cwsw, mxcsr;

        __fenv_getcwsw(&cwsw);
        /* clear reserved bits for no particularly good reason */
        cwsw &= ~0xe0c00000u;

#endif /* ! codereview */
        if (test_sse_hw) {
                /*
                 * pick up exception flags (excluding denormal operand flag)
                 * from mxcsr
                 */
```

```
                /* pick up exception flags (excluding denormal operand
                   flag) from mxcsr */
                __fenv_getmxcsr(&mxcsr);
                cwsw |= (mxcsr & 0x3d);
        }

#endif /* ! codereview */
        cwsw |= *__fex_accrued();
        *fsr = cwsw ^ 0x003f0000u;
}

void
__fenv_setfsr(const unsigned long *fsr)
{
        unsigned int cwsw, mxcsr;
        int te;

        /* save accrued exception flags corresponding to enabled exceptions */
        cwsw = (unsigned int)*fsr;
        te = __fenv_get_te(cwsw);
        *__fex_accrued() = cwsw & te;
        cwsw = (cwsw & ~te) ^ 0x003f0000;

#endif /* ! codereview */
        if (test_sse_hw) {
                /*
                 * propagate rounding direction, masks, and exception flags
                 * (excluding denormal operand mask and flag) to mxcsr
                 */
                /* propagate rounding direction, masks, and exception flags
                   (excluding denormal operand mask and flag) to mxcsr */
                __fenv_getmxcsr(&mxcsr);
                mxcsr = (mxcsr & ~0x7ebd) | ((cwsw >> 13) & 0x6000) | ((cwsw >>
                    9) & 0x1e80) | (cwsw & 0x3d);
                mxcsr = (mxcsr & ~0x7ebd) | ((cwsw >> 13) & 0x6000) |
                        ((cwsw >> 9) & 0x1e80) | (cwsw & 0x3d);
                __fenv_setmxcsr(&mxcsr);
        }

#endif /* ! codereview */
        __fenv_setcwsw(&cwsw);
}

/* Offsets into the fp environment save area (assumes 32-bit protected mode) */
#define CW                      0       /* control word */
#define SW                      1       /* status word */
#define TW                      2       /* tag word */
#define IP                      3       /* instruction pointer */
#define OP                      4       /* opcode */
#define EA                      5       /* operand address */

/* macro for accessing fp registers in the save area */
#if defined(__amd64)
#define fpreg(u, x)             *(long double *)(10 * (x) + \
        (char *)&(u)->uc_mcontext.fpregs.fp_reg_set.fpchip_state.st)
#define fpreg(u,x)      *(long double *)(10*(x)+(char*)&(u)->uc_mcontext.fpregs.
#else
#define fpreg(u, x)             *(long double *)(10 * (x) + \
        (char *)&(u)->uc_mcontext.fpregs.fp_reg_set.fpchip_state.state[7])
#define fpreg(u,x)      *(long double *)(10*(x)+(char*)&(u)->uc_mcontext.fpregs.
#endif

/*
 *  Fix sip->si_code; the Solaris x86 kernel can get it wrong
 */
*  Fix sip->si_code; the Solaris x86 kernel can get it wrong
```



```
*/
void
__fex_get_x86_exc(siginfo_t *sip, ucontext_t *uap)
{
        unsigned sw, cw;

        sw = uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state.status;
#if defined(__amd64)
        cw = uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state.cw;
#else
        cw = uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state.state[CW];
#endif

#endif /* ! codereview */
        if ((sw & FE_INVALID) && !(cw & (1 << fp_trap_invalid)))
                /* store 0 for stack fault, FPE_FLTINV for IEEE invalid op */
                sip->si_code = ((sw & 0x40) ? 0 : FPE_FLTINV);
                sip->si_code = ((sw & 0x40)? 0 : FPE_FLTINV);
        else if ((sw & FE_DIVBYZERO) && !(cw & (1 << fp_trap_division)))
                sip->si_code = FPE_FLTDIV;
        else if ((sw & FE_OVERFLOW) && !(cw & (1 << fp_trap_overflow)))
                sip->si_code = FPE_FLTOVF;
        else if ((sw & FE_UNDERFLOW) && !(cw & (1 << fp_trap_underflow)))
                sip->si_code = FPE_FLTUND;
        else if ((sw & FE_INEXACT) && !(cw & (1 << fp_trap_inexact)))
                sip->si_code = FPE_FLTRES;
        else
                sip->si_code = 0;
}

static enum fp_class_type
my_fp_classf(float *x)
{
        int i = *(int *)x & ~0x80000000;
        int             i = *(int*)x & ~0x80000000;

        if (i < 0x7f800000) {
                if (i < 0x00800000)
                        return ((i == 0) ? fp_zero : fp_subnormal);

                return (fp_normal);
        } else if (i == 0x7f800000) {
                return (fp_infinity);
        } else if (i & 0x400000) {
                return (fp_quiet);
        } else {
                return (fp_signaling);
                        return ((i == 0)? fp_zero : fp_subnormal);
                return fp_normal;
        }
        else if (i == 0x7f800000)
                return fp_infinity;
        else if (i & 0x400000)
                return fp_quiet;
        else
                return fp_signaling;
}

static enum fp_class_type
my_fp_class(double *x)
{
        int i = *(1 + (int *)x) & ~0x80000000;
        int             i = *(1+(int*)x) & ~0x80000000;

        if (i < 0x7ff00000) {
                if (i < 0x00100000)
```

```
229                         return (((i | *(int *)x) == 0) ? fp_zero :
230                             fp_subnormal);

232                 return (fp_normal);
233         } else if (i == 0x7ff00000 && *(int *)x == 0) {
234                 return (fp_infinity);
235         } else if (i & 0x80000) {
236                 return (fp_quiet);
237         } else {
238                 return (fp_signaling);
125                         return (((i | *(int*)x) == 0)? fp_zero : fp_subnormal);
126                 return fp_normal;
239         }
128         else if (i == 0x7ff00000 && *(int*)x == 0)
129                 return fp_infinity;
130         else if (i & 0x80000)
131                 return fp_quiet;
132         else
133                 return fp_signaling;
240 }

242 static enum fp_class_type
243 my_fp_classl(long double *x)
244 {
245         int i = *(2 + (int *)x) & 0x7fff;
139         int             i = *(2+(int*)x) & 0x7fff;

247         if (i < 0x7fff) {
248                 if (i < 1) {
249                         if (*(1 + (int *)x) < 0)
250                                 return (fp_normal);     /* pseudo-denormal */

252                         return (((*(1 + (int *)x) | *(int *)x) == 0) ? fp_zero :
253                             fp_subnormal);
254                 }

256                 return ((*(1 + (int *)x) < 0) ? fp_normal :
257                     (enum fp_class_type)-1);    /* unsupported format */
258         } else if (*(1 + (int *)x) == 0x80000000 && *(int *)x == 0) {
259                 return (fp_infinity);
260         } else if (*(1 + (unsigned *)x) >= 0xc0000000) {
261                 return (fp_quiet);
262         } else if (*(1 + (int *)x) < 0) {
263                 return (fp_signaling);
264         } else {
265                 return ((enum fp_class_type)-1);        /* unsupported format */
266         }
143                         if (*(1+(int*)x) < 0) return fp_normal; /* pseudo-denorm
144                         return (((*(1+(int*)x) | *(int*)x) == 0)?
145                                 fp_zero : fp_subnormal);
146                 }
147                 return ((*(1+(int*)x) < 0)? fp_normal :
148                         (enum fp_class_type) -1); /* unsupported format */
149         }
150         else if (*(1+(int*)x) == 0x80000000 && *(int*)x == 0)
151                 return fp_infinity;
152         else if (*(1+(unsigned*)x) >= 0xc0000000)
153                 return fp_quiet;
154         else if (*(1+(int*)x) < 0)
155                 return fp_signaling;
156         else
157                 return (enum fp_class_type) -1; /* unsupported format */
267 }

269 /*
270  *  Determine which type of invalid operation exception occurred
```



```
271  */
161 *  Determine which type of invalid operation exception occurred
162 */
272 enum fex_exception
273 __fex_get_invalid_type(siginfo_t *sip, ucontext_t *uap)
274 {
275         unsigned op;
276         unsigned long ea;
277         enum fp_class_type t1, t2;

279         /* get the opcode and data address */
280 #if defined(__amd64)
281         op = uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state.fop >> 16;
282         ea = uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state.rdp;
283 #else
284         op = uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state.state[OP] >> 16;
285         ea = uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state.state[EA];
286 #endif

288         /*
289          * if the instruction is fld, the source must be snan (it can't be
290          * an unsupported format, since fldt doesn't raise any exceptions)
291          */
179         /* if the instruction is fld, the source must be snan (it can't be
180            an unsupported format, since fldt doesn't raise any exceptions) */
292         switch (op & 0x7f8) {
293         case 0x100:
294         case 0x140:
295         case 0x180:
296         case 0x500:
297         case 0x540:
298         case 0x580:
299                 return (fex_inv_snan);
188                 return fex_inv_snan;
300         }

302         /* otherwise st is one of the operands; see if it's snan */
303         t1 = my_fp_classl(&fpreg(uap, 0));

305 #endif /* ! codereview */
306         if (t1 == fp_signaling)
307                 return (fex_inv_snan);
308         else if (t1 == (enum fp_class_type)-1)
309                 return ((enum fex_exception)-1);
193                 return fex_inv_snan;
194         else if (t1 == (enum fp_class_type) -1)
195                 return (enum fex_exception) -1;

311         /* determine the class of the second operand if there is one */
312         t2 = fp_normal;

314 #endif /* ! codereview */
315         switch (op & 0x7e0) {
316         case 0x600:
317         case 0x620:
318         case 0x640:
319         case 0x660:
320         case 0x680:
321         case 0x6a0:

323 #endif /* ! codereview */
324                 /* short memory operand */
325                 if (!ea)
326                         return ((enum fex_exception)-1);

199                         return (enum fex_exception) -1;
```

```
                if (*(short *)ea == 0)
                        t2 = fp_zero;

#endif /* ! codereview */
                break;

        case 0x200:
        case 0x220:
        case 0x240:
        case 0x260:
        case 0x280:
        case 0x2a0:

#endif /* ! codereview */
                /* int memory operand */
                if (!ea)
                        return ((enum fex_exception)-1);

                        return (enum fex_exception) -1;
                if (*(int *)ea == 0)
                        t2 = fp_zero;

#endif /* ! codereview */
                break;

        case 0x000:
        case 0x020:
        case 0x040:
        case 0x060:
        case 0x080:
        case 0x0a0:

#endif /* ! codereview */
                /* single precision memory operand */
                if (!ea)
                        return ((enum fex_exception)-1);

                        return (enum fex_exception) -1;
                t2 = my_fp_classf((float *)ea);
                break;

        case 0x400:
        case 0x420:
        case 0x440:
        case 0x460:
        case 0x480:
        case 0x4a0:

#endif /* ! codereview */
                /* double precision memory operand */
                if (!ea)
                        return ((enum fex_exception)-1);

                        return (enum fex_exception) -1;
                t2 = my_fp_class((double *)ea);
                break;

        case 0x0c0:
        case 0x0e0:
        case 0x3e0:
        case 0x4c0:
        case 0x4e0:
        case 0x5e0:
        case 0x6c0:
        case 0x6e0:
        case 0x7e0:
```



```
#endif /* ! codereview */
                /* register operand determined by opcode */
                switch (op & 0x7f8) {
                case 0x3e0:
                case 0x3f8:
                case 0x5f0:
                case 0x5f8:
                case 0x7e0:
                case 0x7f8:
                        /* weed out nonexistent opcodes */
                        break;

                default:
                        t2 = my_fp_classl(&fpreg(uap, op & 7));
                }

#endif /* ! codereview */
                break;

        case 0x1e0:
        case 0x2e0:

#endif /* ! codereview */
                /* special forms */
                switch (op) {
                case 0x1f1:             /* fyl2x */
                case 0x1f3:             /* fpatan */
                case 0x1f5:             /* fprem1 */
                case 0x1f8:             /* fprem */
                case 0x1f9:             /* fyl2xp1 */
                case 0x1fd:             /* fscale */
                case 0x2e9:             /* fucompp */
                        t2 = my_fp_classl(&fpreg(uap, 1));
                        break;
                }

#endif /* ! codereview */
                break;
        }

        /* see if the second op is snan */
        if (t2 == fp_signaling)
                return (fex_inv_snan);
        else if (t2 == (enum fp_class_type)-1)
                return ((enum fex_exception)-1);
                return fex_inv_snan;
        else if (t2 == (enum fp_class_type) -1)
                return (enum fex_exception) -1;

        /* determine the type of operation */
        switch (op & 0x7f8) {
        case 0x000:
        case 0x020:
        case 0x028:
        case 0x040:
        case 0x060:
        case 0x068:
        case 0x080:
        case 0x0a0:
        case 0x0a8:
        case 0x0c0:
        case 0x0e0:
        case 0x0e8:
        case 0x400:
        case 0x420:
```

```
        case 0x428:
        case 0x440:
        case 0x460:
        case 0x468:
        case 0x480:
        case 0x4a0:
        case 0x4a8:
        case 0x4c0:
        case 0x4e0:
        case 0x4e8:
        case 0x6c0:
        case 0x6e0:
        case 0x6e8:

#endif /* ! codereview */
                /* fadd, fsub, fsubr */
                if (t1 == fp_infinity && t2 == fp_infinity)
                        return (fex_inv_isi);

                        return fex_inv_isi;
                break;

        case 0x008:
        case 0x048:
        case 0x088:
        case 0x0c8:
        case 0x208:
        case 0x248:
        case 0x288:
        case 0x408:
        case 0x448:
        case 0x488:
        case 0x4c8:
        case 0x608:
        case 0x648:
        case 0x688:
        case 0x6c8:

#endif /* ! codereview */
                /* fmul */
                if ((t1 == fp_zero && t2 == fp_infinity) || (t2 == fp_zero &&
                    t1 == fp_infinity))
                        return (fex_inv_zmi);

                        return fex_inv_zmi;
                break;

        case 0x030:
        case 0x038:
        case 0x070:
        case 0x078:
        case 0x0b0:
        case 0x0b8:
        case 0x0f0:
        case 0x0f8:
        case 0x230:
        case 0x238:
        case 0x270:
        case 0x278:
        case 0x2b0:
        case 0x2b8:
        case 0x430:
        case 0x438:
        case 0x470:
        case 0x478:
        case 0x4b0:
```



```
        case 0x4b8:
        case 0x4f0:
        case 0x4f8:
        case 0x630:
        case 0x638:
        case 0x670:
        case 0x678:
        case 0x6b0:
        case 0x6b8:
        case 0x6f0:
        case 0x6f8:

#endif /* ! codereview */
                /* fdiv */
                if (t1 == fp_zero && t2 == fp_zero)
                        return (fex_inv_zdz);
                        return fex_inv_zdz;
                else if (t1 == fp_infinity && t2 == fp_infinity)
                        return (fex_inv_idi);

                        return fex_inv_idi;
                break;

        case 0x1f0:
        case 0x1f8:
                /* fsqrt, other special ops */
                return (fex_inv_sqrt);
                return fex_inv_sqrt;

        case 0x010:
        case 0x018:
        case 0x050:
        case 0x058:
        case 0x090:
        case 0x098:
        case 0x0d0:
        case 0x0d8:
        case 0x210:
        case 0x218:
        case 0x250:
        case 0x258:
        case 0x290:
        case 0x298:
        case 0x2e8:
        case 0x3f0:
        case 0x410:
        case 0x418:
        case 0x450:
        case 0x458:
        case 0x490:
        case 0x498:
        case 0x4d0:
        case 0x4d8:
        case 0x5e0:
        case 0x5e8:
        case 0x610:
        case 0x618:
        case 0x650:
        case 0x658:
        case 0x690:
        case 0x698:
        case 0x6d0:
        case 0x6d8:
        case 0x7f0:

#endif /* ! codereview */
```

```
                /* fcom */
                if (t1 == fp_quiet || t2 == fp_quiet)
                        return (fex_inv_cmp);

                        return fex_inv_cmp;
                break;

        case 0x1e0:

#endif /* ! codereview */
                /* ftst */
                if (op == 0x1e4 && t1 == fp_quiet)
                        return (fex_inv_cmp);

                        return fex_inv_cmp;
                break;

        case 0x310:
        case 0x318:
        case 0x350:
        case 0x358:
        case 0x390:
        case 0x398:
        case 0x710:
        case 0x718:
        case 0x730:
        case 0x738:
        case 0x750:
        case 0x758:
        case 0x770:
        case 0x778:
        case 0x790:
        case 0x798:
        case 0x7b0:
        case 0x7b8:
                /* fist, fbst */
                return (fex_inv_int);
                return fex_inv_int;
        }

        return ((enum fex_exception)-1);
        return (enum fex_exception) -1;
}

/* scale factors for exponent unwrapping */
/* 2^12288 */
static const long double
    two12288 = 1.13916522526304337084593857931593200 9e+3699l;
/* 2^-12288 */
static const long double
    twom12288 = 8.778357852076208839765066529179033145e-3700l;
/* (")*(1-2^-64) */
#endif /* ! codereview */
static const long double
        two12288 = 1.139165225263043370845938579315932009e+3699l,       /* 2^122
        twom12288 = 8.778357852076208839765066529179033145e-3700l,      /* 2^-12
    twom12288mulp = 8.778357852076208839289190796475222545e-3700l;
                /* (")*(1-2^-64) */

/* inline templates */
extern long double f2xm1(long double);
extern long double fyl2x(long double, long double);
extern long double fptan(long double);
extern long double fpatan(long double, long double);
extern long double fxtract(long double);
extern long double fprem1(long double, long double);
```



```
extern long double fprem(long double, long double);
extern long double fyl2xp1(long double, long double);
extern long double fsqrt(long double);
extern long double fsincos(long double);
extern long double frndint(long double);
extern long double fscale(long double, long double);
extern long double fsin(long double);
extern long double fcos(long double);

/*
 *  Get the operands, generate the default untrapped result with
 *  exceptions, and set a code indicating the type of operation
 */
 *  Get the operands, generate the default untrapped result with
 *  exceptions, and set a code indicating the type of operation
 */
void
__fex_get_op(siginfo_t *sip, ucontext_t *uap, fex_info_t *info)
{
        fex_numeric_t t;
        long double op2v, x;
        unsigned int cwsw, ex, sw, op;
        unsigned long ea;
        volatile int c __unused;

        /* get the exception type, status word, opcode, and data address */
        ex = sip->si_code;
        sw = uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state.status;
#if defined(__amd64)
        op = uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state.fop >> 16;
        ea = uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state.rdp;
#else
        op = uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state.state[OP] >> 16;
        ea = uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state.state[EA];
#endif

        /*
         * initialize res to the default untrapped result and ex to the
         * corresponding flags (assume trapping is disabled and flags are
         * clear)
         */
        /* initialize res to the default untrapped result and ex to the
           corresponding flags (assume trapping is disabled and flags
           are clear) */

        /* single operand instructions */
        info->op = fex_cnvt;
        info->op2.type = fex_nodata;

#endif /* ! codereview */
        switch (op & 0x7f8) {
        /* load instructions */
        case 0x100:
        case 0x140:
        case 0x180:

#endif /* ! codereview */
                if (!ea) {
                        info->op = fex_other;
                        info->op1.type = info->op2.type = info->res.type =
                            fex_nodata;
                        info->op1.type = info->op2.type = info->res.type = fex_n
                        info->flags = 0;
                        return;
                }
```

```
699 #endif /* ! codereview */
700                 info->op1.type = fex_float;
701                 info->op1.val.f = *(float *)ea;
702                 info->res.type = fex_ldouble;
703                 info->res.val.q = (long double)info->op1.val.f;
446                 info->res.val.q = (long double) info->op1.val.f;
704                 goto done;

706         case 0x500:
707         case 0x540:
708         case 0x580:

710 #endif /* ! codereview */
711                 if (!ea) {
712                         info->op = fex_other;
713                         info->op1.type = info->op2.type = info->res.type =
714                             fex_nodata;
452                         info->op1.type = info->op2.type = info->res.type = fex_n
715                         info->flags = 0;
716                         return;
717                 }

719 #endif /* ! codereview */
720                 info->op1.type = fex_double;
721                 info->op1.val.d = *(double *)ea;
722                 info->res.type = fex_ldouble;
723                 info->res.val.q = (long double)info->op1.val.d;
456                 info->res.val.q = (long double) info->op1.val.d;
724                 goto done;

726         /* store instructions */
727         case 0x110:
728         case 0x118:
729         case 0x150:
730         case 0x158:
731         case 0x190:
732         case 0x198:
733                 info->res.type = fex_float;

735 #endif /* ! codereview */
736                 if (ex == FPE_FLTRES && (op & 8) != 0) {
737                         /* inexact, stack popped */
738                         if (!ea) {
739                                 info->op = fex_other;
740                                 info->op1.type = info->op2.type =
741                                     info->res.type = fex_nodata;
467                                 info->op1.type = info->op2.type = info->res.type
742                                 info->flags = 0;
743                                 return;
744                         }

746 #endif /* ! codereview */
747                         info->op1.type = fex_nodata;
748                         info->res.val.f = *(float *)ea;
749                         info->flags = FE_INEXACT;
750                         return;
751                 }

753 #endif /* ! codereview */
754                 info->op1.type = fex_ldouble;
755                 info->op1.val.q = fpreg(uap, 0);
756                 info->res.val.f = (float)info->op1.val.q;
471                 info->res.val.f = (float) info->op1.val.q;
757                 goto done;

759         case 0x310:
```



```
760         case 0x318:
761         case 0x350:
762         case 0x358:
763         case 0x390:
764         case 0x398:
765                 info->res.type = fex_int;

767 #endif /* ! codereview */
768                 if (ex == FPE_FLTRES && (op & 8) != 0) {
769                         /* inexact, stack popped */
770                         if (!ea) {
771                                 info->op = fex_other;
772                                 info->op1.type = info->op2.type =
773                                     info->res.type = fex_nodata;
481                                 info->op1.type = info->op2.type = info->res.type
774                                 info->flags = 0;
775                                 return;
776                         }

778 #endif /* ! codereview */
779                         info->op1.type = fex_nodata;
780                         info->res.val.i = *(int *)ea;
781                         info->flags = FE_INEXACT;
782                         return;
783                 }

785 #endif /* ! codereview */
786                 info->op1.type = fex_ldouble;
787                 info->op1.val.q = fpreg(uap, 0);
788                 info->res.val.i = (int)info->op1.val.q;
485                 info->res.val.i = (int) info->op1.val.q;
789                 goto done;

791         case 0x510:
792         case 0x518:
793         case 0x550:
794         case 0x558:
795         case 0x590:
796         case 0x598:
797                 info->res.type = fex_double;

799 #endif /* ! codereview */
800                 if (ex == FPE_FLTRES && (op & 8) != 0) {
801                         /* inexact, stack popped */
802                         if (!ea) {
803                                 info->op = fex_other;
804                                 info->op1.type = info->op2.type =
805                                     info->res.type = fex_nodata;
495                                 info->op1.type = info->op2.type = info->res.type
806                                 info->flags = 0;
807                                 return;
808                         }

810 #endif /* ! codereview */
811                         info->op1.type = fex_nodata;
812                         info->res.val.d = *(double *)ea;
813                         info->flags = FE_INEXACT;
814                         return;
815                 }

817 #endif /* ! codereview */
818                 info->op1.type = fex_ldouble;
819                 info->op1.val.q = fpreg(uap, 0);
820                 info->res.val.d = (double)info->op1.val.q;
499                 info->res.val.d = (double) info->op1.val.q;
821                 goto done;
```

```
823         case 0x710:
824         case 0x718:
825         case 0x750:
826         case 0x758:
827         case 0x790:
828         case 0x798:
829                 info->res.type = fex_int;

831 #endif /* ! codereview */
832                 if (ex == FPE_FLTRES && (op & 8) != 0) {
833                         /* inexact, stack popped */
834                         if (!ea) {
835                                 info->op = fex_other;
836                                 info->op1.type = info->op2.type =
837                                     info->res.type = fex_nodata;
509                                 info->op1.type = info->op2.type = info->res.type
838                                 info->flags = 0;
839                                 return;
840                         }

842 #endif /* ! codereview */
843                         info->op1.type = fex_nodata;
844                         info->res.val.i = *(short *)ea;
845                         info->flags = FE_INEXACT;
846                         return;
847                 }

849 #endif /* ! codereview */
850                 info->op1.type = fex_ldouble;
851                 info->op1.val.q = fpreg(uap, 0);
852                 info->res.val.i = (short)info->op1.val.q;
513                 info->res.val.i = (short) info->op1.val.q;
853                 goto done;

855         case 0x730:
856         case 0x770:
857         case 0x7b0:
858                 /* fbstp; don't bother */
859                 info->op = fex_other;
860                 info->op1.type = info->res.type = fex_nodata;
861                 info->flags = 0;
862                 return;

864         case 0x738:
865         case 0x778:
866         case 0x7b8:
867                 info->res.type = fex_llong;

869 #endif /* ! codereview */
870                 if (ex == FPE_FLTRES) {
871                         /* inexact, stack popped */
872                         if (!ea) {
873                                 info->op = fex_other;
874                                 info->op1.type = info->op2.type =
875                                     info->res.type = fex_nodata;
529                                 info->op1.type = info->op2.type = info->res.type
876                                 info->flags = 0;
877                                 return;
878                         }

880 #endif /* ! codereview */
881                         info->op1.type = fex_nodata;
882                         info->res.val.l = *(long long *)ea;
883                         info->flags = FE_INEXACT;
884                         return;
```



```
885                 }

887 #endif /* ! codereview */
888                 info->op1.type = fex_ldouble;
889                 info->op1.val.q = fpreg(uap, 0);
890                 info->res.val.l = (long long)info->op1.val.q;
533                 info->res.val.l = (long long) info->op1.val.q;
891                 goto done;
892         }

894         /*
895          * all other ops (except compares) have destinations on the stack so
896          * overflow, underflow, and inexact will stomp their operands
897          */
537         /* all other ops (except compares) have destinations on the stack
538            so overflow, underflow, and inexact will stomp their operands */
898         if (ex == FPE_FLTOVF || ex == FPE_FLTUND || ex == FPE_FLTRES) {
899                 /* find the trapped result */
900                 info->op1.type = info->op2.type = fex_nodata;
901                 info->res.type = fex_ldouble;

903 #endif /* ! codereview */
904                 switch (op & 0x7f8) {
905                 case 0x1f0:
906                         /* fptan pushes 1.0 afterward, so result is in st(1) */
907                         info->res.val.q = ((op == 0x1f2) ? fpreg(uap, 1) :
543                         info->res.val.q = ((op == 0x1f2)? fpreg(uap, 1) :
908                             fpreg(uap, 0));
909                         break;

911                 case 0x4c0:
912                 case 0x4c8:
913                 case 0x4e0:
914                 case 0x4e8:
915                 case 0x4f0:
916                 case 0x4f8:
917                         info->res.val.q = fpreg(uap, op & 7);
918                         break;

920                 case 0x6c0:
921                 case 0x6c8:
922                 case 0x6e0:
923                 case 0x6e8:
924                 case 0x6f0:
925                 case 0x6f8:
926                         /* stack was popped afterward */
927                         info->res.val.q = fpreg(uap, (op - 1) & 7);
928                         break;

930                 default:
931                         info->res.val.q = fpreg(uap, 0);
932                 }

934                 /* reconstruct default untrapped result */
935                 if (ex == FPE_FLTOVF) {
936                         /* generate an overflow with the sign of the result */
937                         x = two12288;
938                         *(4 + (short *)&x) |= (*(4 +
939                             (short *)&info->res.val.q) & 0x8000);
574                         *(4+(short*)&x) |= (*(4+(short*)&info->res.val.q) & 0x80
940                         info->res.val.q = x * two12288;
941                         info->flags = FE_OVERFLOW | FE_INEXACT;
942                         __fenv_getcwsw(&cwsw);
943                         cwsw &= ~FE_ALL_EXCEPT;
944                         __fenv_setcwsw(&cwsw);
945                 } else if (ex == FPE_FLTUND) {
```

```
946                         /*
947                          * undo the scaling; we can't distinguish a chopped
948                          * result from an exact one without futzing around to
949                          * trap all in- exact exceptions so as to keep the
950                          * flag clear, so we just punt
951                          */
580                 }
581                 else if (ex == FPE_FLTUND) {
582                         /* undo the scaling; we can't distinguish a chopped resu
583                            from an exact one without futzing around to trap all
584                            exact exceptions so as to keep the flag clear, so we
585                            punt */
952                         if (sw & 0x200) /* result was rounded up */
953                                 info->res.val.q = (info->res.val.q *
954                                     twom12288) * twom12288mulp;
587                                 info->res.val.q = (info->res.val.q * twom12288)
955                         else
956                                 info->res.val.q = (info->res.val.q *
957                                     twom12288) * twom12288;

589                                 info->res.val.q = (info->res.val.q * twom12288)
959                         __fenv_getcwsw(&cwsw);
960                         info->flags = (cwsw & FE_INEXACT) | FE_UNDERFLOW;
961                         cwsw &= ~FE_ALL_EXCEPT;
962                         __fenv_setcwsw(&cwsw);
963                 } else {
964                         info->flags = FE_INEXACT;
965 #endif /* ! codereview */
966                 }
594                 else
595                         info->flags = FE_INEXACT;

968                 /* determine the operation code */
969                 switch (op) {
970                 case 0x1f0:             /* f2xm1 */
971                 case 0x1f1:             /* fyl2x */
972                 case 0x1f2:             /* fptan */
973                 case 0x1f3:             /* fpatan */
974                 case 0x1f5:             /* fprem1 */
975                 case 0x1f8:             /* fprem */
976                 case 0x1f9:             /* fyl2xp1 */
977                 case 0x1fb:             /* fsincos */
978                 case 0x1fc:             /* frndint */
979                 case 0x1fd:             /* fscale */
980                 case 0x1fe:             /* fsin */
981                 case 0x1ff:             /* fcos */
982                         info->op = fex_other;
983                         return;

985                 case 0x1fa:             /* fsqrt */
986                         info->op = fex_sqrt;
987                         return;
988                 }

990                 info->op = fex_other;

992 #endif /* ! codereview */
993                 switch (op & 0x7c0) {
994                 case 0x000:
995                 case 0x040:
996                 case 0x080:
997                 case 0x0c0:
998                 case 0x200:
999                 case 0x240:
1000                case 0x280:
1001                case 0x400:
```



```
1002                case 0x440:
1003                case 0x480:
1004                case 0x4c0:
1005                case 0x600:
1006                case 0x640:
1007                case 0x680:
1008                case 0x6c0:

1010 #endif /* ! codereview */
1011                        switch (op & 0x38) {
1012                        case 0x00:
1013                                info->op = fex_add;
1014                                break;

1016                        case 0x08:
1017                                info->op = fex_mul;
1018                                break;

1020                        case 0x20:
1021                        case 0x28:
1022                                info->op = fex_sub;
1023                                break;

1025                        case 0x30:
1026                        case 0x38:
1027                                info->op = fex_div;
1028                                break;
1029                        }
1030                }

1032 #endif /* ! codereview */
1033                return;
1034        }

1036        /*
1037         * for other exceptions, the operands are preserved, so we can just
1038         * emulate the operation with traps disabled
1039         */
620         /* for other exceptions, the operands are preserved, so we can
621            just emulate the operation with traps disabled */

1041        /* one operand is always in st */
1042        info->op1.type = fex_ldouble;
1043        info->op1.val.q = fpreg(uap, 0);

1045        /* oddball instructions */
1046        info->op = fex_other;

1048 #endif /* ! codereview */
1049        switch (op) {
1050        case 0x1e4:                     /* ftst */
1051                info->op = fex_cmp;
1052                info->op2.type = fex_ldouble;
1053                info->op2.val.q = 0.0l;
1054                info->res.type = fex_nodata;
1055                c = (info->op1.val.q < info->op2.val.q);
1056                goto done;

1058        case 0x1f0:                     /* f2xm1 */
1059                info->res.type = fex_ldouble;
1060                info->res.val.q = f2xm1(info->op1.val.q);
1061                goto done;

1063        case 0x1f1:                     /* fyl2x */
1064                info->op2.type = fex_ldouble;
1065                info->op2.val.q = fpreg(uap, 1);
```

```
                info->res.type = fex_ldouble;
                info->res.val.q = fyl2x(info->op1.val.q, info->op2.val.q);
                goto done;

        case 0x1f2:                     /* fptan */
                info->res.type = fex_ldouble;
                info->res.val.q = fptan(info->op1.val.q);
                goto done;

        case 0x1f3:                     /* fpatan */
                info->op2.type = fex_ldouble;
                info->op2.val.q = fpreg(uap, 1);
                info->res.type = fex_ldouble;
                info->res.val.q = fpatan(info->op1.val.q, info->op2.val.q);
                goto done;

        case 0x1f4:                     /* fxtract */
                info->res.type = fex_ldouble;
                info->res.val.q = fxtract(info->op1.val.q);
                goto done;

        case 0x1f5:                     /* fprem1 */
                info->op2.type = fex_ldouble;
                info->op2.val.q = fpreg(uap, 1);
                info->res.type = fex_ldouble;
                info->res.val.q = fprem1(info->op1.val.q, info->op2.val.q);
                goto done;

        case 0x1f8:                     /* fprem */
                info->op2.type = fex_ldouble;
                info->op2.val.q = fpreg(uap, 1);
                info->res.type = fex_ldouble;
                info->res.val.q = fprem(info->op1.val.q, info->op2.val.q);
                goto done;

        case 0x1f9:                     /* fyl2xp1 */
                info->op2.type = fex_ldouble;
                info->op2.val.q = fpreg(uap, 1);
                info->res.type = fex_ldouble;
                info->res.val.q = fyl2xp1(info->op1.val.q, info->op2.val.q);
                goto done;

        case 0x1fa:                     /* fsqrt */
                info->op = fex_sqrt;
                info->res.type = fex_ldouble;
                info->res.val.q = fsqrt(info->op1.val.q);
                goto done;

        case 0x1fb:                     /* fsincos */
                info->res.type = fex_ldouble;
                info->res.val.q = fsincos(info->op1.val.q);
                goto done;

        case 0x1fc:                     /* frndint */
                info->res.type = fex_ldouble;
                info->res.val.q = frndint(info->op1.val.q);
                goto done;

        case 0x1fd:                     /* fscale */
                info->op2.type = fex_ldouble;
                info->op2.val.q = fpreg(uap, 1);
                info->res.type = fex_ldouble;
                info->res.val.q = fscale(info->op1.val.q, info->op2.val.q);
                goto done;

        case 0x1fe:                     /* fsin */
```



```
                info->res.type = fex_ldouble;
                info->res.val.q = fsin(info->op1.val.q);
                goto done;

        case 0x1ff:                     /* fcos */
                info->res.type = fex_ldouble;
                info->res.val.q = fcos(info->op1.val.q);
                goto done;

        case 0x2e9:                     /* fucompp */
                info->op = fex_cmp;
                info->op2.type = fex_ldouble;
                info->op2.val.q = fpreg(uap, 1);
                info->res.type = fex_nodata;
                c = (info->op1.val.q == info->op2.val.q);
                goto done;
        }

        /* fucom[p], fcomi[p], fucomi[p] */
        switch (op & 0x7f8) {
        case 0x3e8:
        case 0x5e0:
        case 0x5e8:
        case 0x7e8:                     /* unordered compares */
                info->op = fex_cmp;
                info->op2.type = fex_ldouble;
                info->op2.val.q = fpreg(uap, op & 7);
                info->res.type = fex_nodata;
                c = (info->op1.val.q == info->op2.val.q);
                goto done;

        case 0x3f0:
        case 0x7f0:                     /* ordered compares */
                info->op = fex_cmp;
                info->op2.type = fex_ldouble;
                info->op2.val.q = fpreg(uap, op & 7);
                info->res.type = fex_nodata;
                c = (info->op1.val.q < info->op2.val.q);
                goto done;
        }

        /*
         * all other instructions come in groups of the form
         * fadd, fmul, fcom, fcomp, fsub, fsubr, fdiv, fdivr
         */
        /* all other instructions come in groups of the form
           fadd, fmul, fcom, fcomp, fsub, fsubr, fdiv, fdivr */

        /* get the second operand */
        switch (op & 0x7c0) {
        case 0x000:
        case 0x040:
        case 0x080:

#endif /* ! codereview */
                if (!ea) {
                        info->op = fex_other;
                        info->op1.type = info->op2.type = info->res.type =
                            fex_nodata;
                        info->op1.type = info->op2.type = info->res.type = fex_n
                        info->flags = 0;
                        return;
                }

#endif /* ! codereview */
                info->op2.type = fex_float;
```

```
                info->op2.val.f = *(float *)ea;
                op2v = (long double)info->op2.val.f;
 641            op2v = (long double) info->op2.val.f;
                break;

        case 0x0c0:
                info->op2.type = fex_ldouble;
                op2v = info->op2.val.q = fpreg(uap, op & 7);
                break;

        case 0x200:
        case 0x240:
        case 0x280:

#endif /* ! codereview */
                if (!ea) {
                        info->op = fex_other;
                        info->op1.type = info->op2.type = info->res.type =
                            fex_nodata;
 652                    info->op1.type = info->op2.type = info->res.type = fex_n
                        info->flags = 0;
                        return;
                }

#endif /* ! codereview */
                info->op2.type = fex_int;
                info->op2.val.i = *(int *)ea;
                op2v = (long double)info->op2.val.i;
 656            op2v = (long double) info->op2.val.i;
                break;

        case 0x400:
        case 0x440:
        case 0x480:

#endif /* ! codereview */
                if (!ea) {
                        info->op = fex_other;
                        info->op1.type = info->op2.type = info->res.type =
                            fex_nodata;
 662                    info->op1.type = info->op2.type = info->res.type = fex_n
                        info->flags = 0;
                        return;
                }

#endif /* ! codereview */
                info->op2.type = fex_double;
                info->op2.val.d = *(double *)ea;
                op2v = (long double)info->op2.val.d;
 666            op2v = (long double) info->op2.val.d;
                break;

        case 0x4c0:
        case 0x6c0:
                info->op2.type = fex_ldouble;
                info->op2.val.q = fpreg(uap, op & 7);
                t = info->op1;
                info->op1 = info->op2;
                info->op2 = t;
                op2v = info->op2.val.q;
                break;

        case 0x600:
        case 0x640:
        case 0x680:
```



```
#endif /* ! codereview */
                if (!ea) {
                        info->op = fex_other;
                        info->op1.type = info->op2.type = info->res.type =
                            fex_nodata;
 682                    info->op1.type = info->op2.type = info->res.type = fex_n
                        info->flags = 0;
                        return;
                }

#endif /* ! codereview */
                info->op2.type = fex_int;
                info->op2.val.i = *(short *)ea;
                op2v = (long double)info->op2.val.i;
 686            op2v = (long double) info->op2.val.i;
                break;

        default:
                info->op = fex_other;
                info->op1.type = info->op2.type = info->res.type = fex_nodata;
                info->flags = 0;
                return;
        }

        /* distinguish different operations in the group */
        info->res.type = fex_ldouble;

#endif /* ! codereview */
        switch (op & 0x38) {
        case 0x00:
                info->op = fex_add;
                info->res.val.q = info->op1.val.q + op2v;
                break;

        case 0x08:
                info->op = fex_mul;
                info->res.val.q = info->op1.val.q * op2v;
                break;

        case 0x10:
        case 0x18:
                info->op = fex_cmp;
                info->res.type = fex_nodata;
                c = (info->op1.val.q < op2v);
                break;

        case 0x20:
                info->op = fex_sub;
                info->res.val.q = info->op1.val.q - op2v;
                break;

        case 0x28:
                info->op = fex_sub;
                info->res.val.q = op2v - info->op1.val.q;
                t = info->op1;
                info->op1 = info->op2;
                info->op2 = t;
                break;

        case 0x30:
                info->op = fex_div;
                info->res.val.q = info->op1.val.q / op2v;
                break;

        case 0x38:
                info->op = fex_div;
```

```
                info->res.val.q = op2v / info->op1.val.q;
                t = info->op1;
                info->op1 = info->op2;
                info->op2 = t;
                break;

        default:
                info->op = fex_other;
                info->op1.type = info->op2.type = info->res.type = fex_nodata;
                info->flags = 0;
                return;
        }

done:
        __fenv_getcwsw(&cwsw);
        info->flags = cwsw & FE_ALL_EXCEPT;
        cwsw &= ~FE_ALL_EXCEPT;
        __fenv_setcwsw(&cwsw);
}

/* pop the saved stack */
static void
pop(ucontext_t *uap)
 698 static void pop(ucontext_t *uap)
{
        unsigned top;

        fpreg(uap, 0) = fpreg(uap, 1);
        fpreg(uap, 1) = fpreg(uap, 2);
        fpreg(uap, 2) = fpreg(uap, 3);
        fpreg(uap, 3) = fpreg(uap, 4);
        fpreg(uap, 4) = fpreg(uap, 5);
        fpreg(uap, 5) = fpreg(uap, 6);
        fpreg(uap, 6) = fpreg(uap, 7);
#if defined(__amd64)
        top = (uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state.sw >> 10) & 0xe;
 710    top = (uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state.sw >> 10)
 711            & 0xe;
        uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state.fctw |= (3 << top);
        top = (top + 2) & 0xe;
        uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state.sw =
            (uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state.sw & ~0x3800) |
            (top << 10);
 715            (uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state.sw & ~0x3800)
 716            | (top << 10);
#else
        top = (uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state.state[SW] >>
            10) & 0xe;
 718    top = (uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state.state[SW] >> 10)
 719            & 0xe;
        uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state.state[TW] |= (3 << top);
        top = (top + 2) & 0xe;
        uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state.state[SW] =
            (uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state.state[SW] &
            ~0x3800) | (top << 10);
 723            (uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state.state[SW] & ~0x
 724            | (top << 10);
#endif
}

/* push x onto the saved stack */
static void
push(long double x, ucontext_t *uap)
 729 static void push(long double x, ucontext_t *uap)
{
        unsigned top;
```



```
        fpreg(uap, 7) = fpreg(uap, 6);
        fpreg(uap, 6) = fpreg(uap, 5);
        fpreg(uap, 5) = fpreg(uap, 4);
        fpreg(uap, 4) = fpreg(uap, 3);
        fpreg(uap, 3) = fpreg(uap, 2);
        fpreg(uap, 2) = fpreg(uap, 1);
        fpreg(uap, 1) = fpreg(uap, 0);
        fpreg(uap, 0) = x;
#if defined(__amd64)
        top = (uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state.sw >> 10) & 0xe;
 742    top = (uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state.sw >> 10)
 743            & 0xe;
        top = (top - 2) & 0xe;
        uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state.fctw &= ~(3 << top);
        uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state.sw =
            (uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state.sw & ~0x3800) |
            (top << 10);
 747            (uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state.sw & ~0x3800)
 748            | (top << 10);
#else
        top = (uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state.state[SW] >>
            10) & 0xe;
 750    top = (uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state.state[SW] >> 10)
 751            & 0xe;
        top = (top - 2) & 0xe;
        uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state.state[TW] &= ~(3 <<
            top);
 753    uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state.state[TW] &= ~(3 << top)
        uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state.state[SW] =
            (uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state.state[SW] &
            ~0x3800) | (top << 10);
 755            (uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state.state[SW] & ~0x
 756            | (top << 10);
#endif
}

/* scale factors for exponent wrapping */
static const float fun = 7.922816251e+28f,              /* 2^96 */
 761 static const float
 762    fun = 7.922816251e+28f, /* 2^96 */
        fov = 1.262177448e-29f;                         /* 2^-96 */
static const double dun = 1.552518092300708935e+231,    /* 2^768 */
 764 static const double
 765    dun = 1.552518092300708935e+231,        /* 2^768 */
        dov = 6.441148769597133308e-232;                /* 2^-768 */

/*
 *  Store the specified result; if no result is given but the exception
 *  is underflow or overflow, use the default trapped result
 */
 769 *  Store the specified result; if no result is given but the exception
 770 *  is underflow or overflow, use the default trapped result
 771 */
void
__fex_st_result(siginfo_t *sip, ucontext_t *uap, fex_info_t *info)
{
        fex_numeric_t r;
        unsigned long ex, op, ea, stack;

        /* get the exception type, opcode, and data address */
        ex = sip->si_code;
#if defined(__amd64)
        op = uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state.fop >> 16;
        ea = uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state.rdp; /* ??? */
 782    ea = uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state.rdp; /*???*/
```

```
#else
        op = uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state.state[OP] >> 16;
        ea = uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state.state[EA];
#endif

        /*
         * if the instruction is a compare, set the condition codes
         * to unordered and update the stack
         */
        /* if the instruction is a compare, set the condition codes
           to unordered and update the stack */
        switch (op & 0x7f8) {
        case 0x010:
        case 0x050:
        case 0x090:
        case 0x0d0:
        case 0x210:
        case 0x250:
        case 0x290:
        case 0x410:
        case 0x450:
        case 0x490:
        case 0x4d0:
        case 0x5e0:
        case 0x610:
        case 0x650:
        case 0x690:
                /* f[u]com */
#if defined(__amd64)
                uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state.sw |= 0x4500;
#else
                uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state.state[SW] |=
                    0x4500;
                uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state.state[SW] |= 0x4
#endif
                return;

        case 0x018:
        case 0x058:
        case 0x098:
        case 0x0d8:
        case 0x218:
        case 0x258:
        case 0x298:
        case 0x418:
        case 0x458:
        case 0x498:
        case 0x4d8:
        case 0x5e8:
        case 0x618:
        case 0x658:
        case 0x698:
        case 0x6d0:
                /* f[u]comp */
#if defined(__amd64)
                uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state.sw |= 0x4500;
#else
                uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state.state[SW] |=
                    0x4500;
                uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state.state[SW] |= 0x4
#endif
                pop(uap);
                return;

        case 0x2e8:
        case 0x6d8:
```



```
                /* f[u]compp */
#if defined(__amd64)
                uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state.sw |= 0x4500;
#else
                uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state.state[SW] |=
                    0x4500;
                uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state.state[SW] |= 0x4
#endif
                pop(uap);
                pop(uap);
                return;

        case 0x1e0:

#endif /* ! codereview */
                if (op == 0x1e4) {      /* ftst */
#if defined(__amd64)
                        uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state.sw |=
                            0x4500;
                        uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state.sw |= 0x
#else
                        uap->uc_mcontext.fpregs.fp_reg_set.
                            fpchip_state.state[SW] |= 0x4500;
                        uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state.state[SW
#endif
                        return;
                }

#endif /* ! codereview */
                break;

        case 0x3e8:
        case 0x3f0:
                /* f[u]comi */
#if defined(__amd64)
                uap->uc_mcontext.gregs[REG_PS] |= 0x45;
#else
                uap->uc_mcontext.gregs[EFL] |= 0x45;
#endif
                return;

        case 0x7e8:
        case 0x7f0:
                /* f[u]comip */
#if defined(__amd64)
                uap->uc_mcontext.gregs[REG_PS] |= 0x45;
#else
                uap->uc_mcontext.gregs[EFL] |= 0x45;
#endif
                pop(uap);
                return;
        }

        /*
         * if there is no result available and the exception is overflow
         * or underflow, use the wrapped result
         */
        /* if there is no result available and the exception is overflow
           or underflow, use the wrapped result */
        r = info->res;

#endif /* ! codereview */
        if (r.type == fex_nodata) {
                if (ex == FPE_FLTOVF || ex == FPE_FLTUND) {
                        /* for store instructions, do the scaling and store */
                        switch (op & 0x7f8) {
```

```
                        case 0x110:
                        case 0x118:
                        case 0x150:
                        case 0x158:
                        case 0x190:
                        case 0x198:

#endif /* ! codereview */
                                if (!ea)
                                        return;

#endif /* ! codereview */
                                if (ex == FPE_FLTOVF)
                                        *(float *)ea = (fpreg(uap, 0) * fov) *
                                            fov;
                                        *(float *)ea = (fpreg(uap, 0) * fov) * f
                                else
                                        *(float *)ea = (fpreg(uap, 0) * fun) *
                                            fun;

                                        *(float *)ea = (fpreg(uap, 0) * fun) * f
                                if ((op & 8) != 0)
                                        pop(uap);

#endif /* ! codereview */
                                break;

                        case 0x510:
                        case 0x518:
                        case 0x550:
                        case 0x558:
                        case 0x590:
                        case 0x598:

#endif /* ! codereview */
                                if (!ea)
                                        return;

#endif /* ! codereview */
                                if (ex == FPE_FLTOVF)
                                        *(double *)ea = (fpreg(uap, 0) * dov) *
                                            dov;
                                        *(double *)ea = (fpreg(uap, 0) * dov) *
                                else
                                        *(double *)ea = (fpreg(uap, 0) * dun) *
                                            dun;

                                        *(double *)ea = (fpreg(uap, 0) * dun) *
                                if ((op & 8) != 0)
                                        pop(uap);

#endif /* ! codereview */
                                break;
                        }
                }

#endif /* ! codereview */
#ifdef DEBUG
                else if (ex != FPE_FLTRES) {
                else if (ex != FPE_FLTRES)
                        printf("No result supplied, stack may be hosed\n");
                }
#endif /* ! codereview */
#endif
                return;
        }
```



```
        /*
         * otherwise convert the supplied result to the correct type, put it
         * in the destination, and update the stack as need be
         */
        /* otherwise convert the supplied result to the correct type,
           put it in the destination, and update the stack as need be */

        /* store instructions */
        switch (op & 0x7f8) {
        case 0x110:
        case 0x118:
        case 0x150:
        case 0x158:
        case 0x190:
        case 0x198:

#endif /* ! codereview */
                if (!ea)
                        return;

#endif /* ! codereview */
                switch (r.type) {
                case fex_int:
                        *(float *)ea = (float)r.val.i;
                        *(float *)ea = (float) r.val.i;
                        break;

                case fex_llong:
                        *(float *)ea = (float)r.val.l;
                        *(float *)ea = (float) r.val.l;
                        break;

                case fex_float:
                        *(float *)ea = r.val.f;
                        break;

                case fex_double:
                        *(float *)ea = (float)r.val.d;
                        *(float *)ea = (float) r.val.d;
                        break;

                case fex_ldouble:
                        *(float *)ea = (float)r.val.q;
                        *(float *)ea = (float) r.val.q;
                        break;

                default:
                        break;
                }

#endif /* ! codereview */
                if (ex != FPE_FLTRES && (op & 8) != 0)
                        pop(uap);

#endif /* ! codereview */
                return;

        case 0x310:
        case 0x318:
        case 0x350:
        case 0x358:
        case 0x390:
        case 0x398:

#endif /* ! codereview */
```

```
                if (!ea)
                        return;

#endif /* ! codereview */
                switch (r.type) {
                case fex_int:
                        *(int *)ea = r.val.i;
                        break;

                case fex_llong:
                        *(int *)ea = (int)r.val.l;
                        *(int *)ea = (int) r.val.l;
                        break;

                case fex_float:
                        *(int *)ea = (int)r.val.f;
                        *(int *)ea = (int) r.val.f;
                        break;

                case fex_double:
                        *(int *)ea = (int)r.val.d;
                        *(int *)ea = (int) r.val.d;
                        break;

                case fex_ldouble:
                        *(int *)ea = (int)r.val.q;
                        *(int *)ea = (int) r.val.q;
                        break;

                default:
                        break;
                }

#endif /* ! codereview */
                if (ex != FPE_FLTRES && (op & 8) != 0)
                        pop(uap);

#endif /* ! codereview */
                return;

        case 0x510:
        case 0x518:
        case 0x550:
        case 0x558:
        case 0x590:
        case 0x598:

#endif /* ! codereview */
                if (!ea)
                        return;

#endif /* ! codereview */
                switch (r.type) {
                case fex_int:
                        *(double *)ea = (double)r.val.i;
                        *(double *)ea = (double) r.val.i;
                        break;

                case fex_llong:
                        *(double *)ea = (double)r.val.l;
                        *(double *)ea = (double) r.val.l;
                        break;

                case fex_float:
                        *(double *)ea = (double)r.val.f;
                        *(double *)ea = (double) r.val.f;
```



```
                        break;

                case fex_double:
                        *(double *)ea = r.val.d;
                        break;

                case fex_ldouble:
                        *(double *)ea = (double)r.val.q;
                        *(double *)ea = (double) r.val.q;
                        break;

                default:
                        break;
                }

#endif /* ! codereview */
                if (ex != FPE_FLTRES && (op & 8) != 0)
                        pop(uap);

#endif /* ! codereview */
                return;

        case 0x710:
        case 0x718:
        case 0x750:
        case 0x758:
        case 0x790:
        case 0x798:

#endif /* ! codereview */
                if (!ea)
                        return;

#endif /* ! codereview */
                switch (r.type) {
                case fex_int:
                        *(short *)ea = (short)r.val.i;
                        *(short *)ea = (short) r.val.i;
                        break;

                case fex_llong:
                        *(short *)ea = (short)r.val.l;
                        *(short *)ea = (short) r.val.l;
                        break;

                case fex_float:
                        *(short *)ea = (short)r.val.f;
                        *(short *)ea = (short) r.val.f;
                        break;

                case fex_double:
                        *(short *)ea = (short)r.val.d;
                        *(short *)ea = (short) r.val.d;
                        break;

                case fex_ldouble:
                        *(short *)ea = (short)r.val.q;
                        *(short *)ea = (short) r.val.q;
                        break;

                default:
                        break;
                }

#endif /* ! codereview */
                if (ex != FPE_FLTRES && (op & 8) != 0)
```

```
1788                        pop(uap);

1790 #endif /* ! codereview */
1791                return;

1793        case 0x730:
1794        case 0x770:
1795        case 0x7b0:

1797 #endif /* ! codereview */
1798                /* fbstp; don't bother */
1799                if (ea && ex != FPE_FLTRES)
1800                        pop(uap);

1802 #endif /* ! codereview */
1803                return;

1805        case 0x738:
1806        case 0x778:
1807        case 0x7b8:

1809 #endif /* ! codereview */
1810                if (!ea)
1811                        return;

1813 #endif /* ! codereview */
1814                switch (r.type) {
1815                case fex_int:
1816                        *(long long *)ea = (long long)r.val.i;
 968                        *(long long *)ea = (long long) r.val.i;
1817                        break;

1819                case fex_llong:
1820                        *(long long *)ea = r.val.l;
1821                        break;

1823                case fex_float:
1824                        *(long long *)ea = (long long)r.val.f;
 976                        *(long long *)ea = (long long) r.val.f;
1825                        break;

1827                case fex_double:
1828                        *(long long *)ea = (long long)r.val.d;
 980                        *(long long *)ea = (long long) r.val.d;
1829                        break;

1831                case fex_ldouble:
1832                        *(long long *)ea = (long long)r.val.q;
 984                        *(long long *)ea = (long long) r.val.q;
1833                        break;

1835                default:
1836                        break;
1837                }

1839 #endif /* ! codereview */
1840                if (ex != FPE_FLTRES)
1841                        pop(uap);

1843 #endif /* ! codereview */
1844                return;
1845        }

1847        /* for all other instructions, the result goes into a register */
1848        switch (r.type) {
1849        case fex_int:
```



```
1850                r.val.q = (long double)r.val.i;
 990                r.val.q = (long double) r.val.i;
1851                break;

1853        case fex_llong:
1854                r.val.q = (long double)r.val.l;
 994                r.val.q = (long double) r.val.l;
1855                break;

1857        case fex_float:
1858                r.val.q = (long double)r.val.f;
 998                r.val.q = (long double) r.val.f;
1859                break;

1861        case fex_double:
1862                r.val.q = (long double)r.val.d;
1002                r.val.q = (long double) r.val.d;
1863                break;

1865        default:
1866                break;
1867        }

1869        /* for load instructions, push the result onto the stack */
1870        switch (op & 0x7f8) {
1871        case 0x100:
1872        case 0x140:
1873        case 0x180:
1874        case 0x500:
1875        case 0x540:
1876        case 0x580:

1878 #endif /* ! codereview */
1879                if (ea)
1880                        push(r.val.q, uap);

1882 #endif /* ! codereview */
1883                return;
1884        }

1886        /*
1887         * for all other instructions, if the exception is overflow,
1888         * underflow, or inexact, the stack has already been updated
1889         */
1017        /* for all other instructions, if the exception is overflow,
1018           underflow, or inexact, the stack has already been updated */
1890        stack = (ex == FPE_FLTOVF || ex == FPE_FLTUND || ex == FPE_FLTRES);

1892 #endif /* ! codereview */
1893        switch (op & 0x7f8) {
1894        case 0x1f0:                     /* oddballs */

1896 #endif /* ! codereview */
1897                switch (op) {
1898                case 0x1f1:             /* fyl2x */
1899                case 0x1f3:             /* fpatan */
1900                case 0x1f9:             /* fyl2xp1 */

1902 #endif /* ! codereview */
1903                        /* pop the stack, leaving the result in st */
1904                        if (!stack)
1905                                pop(uap);

1907 #endif /* ! codereview */
1908                        fpreg(uap, 0) = r.val.q;
1909                        return;
```

```
1911                 case 0x1f2:             /* fpatan */

1913 #endif /* ! codereview */
1914                         /* fptan pushes 1.0 afterward */
1915                         if (stack) {
1020                         if (stack)
1916                                 fpreg(uap, 1) = r.val.q;
1917                         } else {
1022                         else {
1918                                 fpreg(uap, 0) = r.val.q;
1919                                 push(1.0L, uap);
1920                         }

1922 #endif /* ! codereview */
1923                         return;

1925                 case 0x1f4:             /* fxtract */
1926                 case 0x1fb:             /* fsincos */

1928 #endif /* ! codereview */
1929                         /* leave the supplied result in st */
1930                         if (stack) {
1026                         if (stack)
1931                                 fpreg(uap, 0) = r.val.q;
1932                         } else {
1028                         else {
1933                                 fpreg(uap, 0) = 0.0;    /* punt */
1934                                 push(r.val.q, uap);
1935                         }

1937 #endif /* ! codereview */
1938                         return;
1939                 }

1941                 /* all others leave the stack alone and the result in st */
1942                 fpreg(uap, 0) = r.val.q;
1943                 return;

1945         case 0x4c0:
1946         case 0x4c8:
1947         case 0x4e0:
1948         case 0x4e8:
1949         case 0x4f0:
1950         case 0x4f8:
1951                 fpreg(uap, op & 7) = r.val.q;
1952                 return;

1954         case 0x6c0:
1955         case 0x6c8:
1956         case 0x6e0:
1957         case 0x6e8:
1958         case 0x6f0:
1959         case 0x6f8:

1961 #endif /* ! codereview */
1962                 /* stack is popped afterward */
1963                 if (stack) {
1032                 if (stack)
1964                         fpreg(uap, (op - 1) & 7) = r.val.q;
1965                 } else {
1034                 else {
1966                         fpreg(uap, op & 7) = r.val.q;
1967                         pop(uap);
1968                 }
```



```
1970 #endif /* ! codereview */
1971                 return;

1973         default:
1974                 fpreg(uap, 0) = r.val.q;
1975                 return;
1976         }
1977 }
```

```
**********************************************************
   21617 Thu Jun  6 20:40:05 2019
new/usr/src/lib/libm/common/m9x/__fex_sparc.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
   1 /*
   2  * CDDL HEADER START
   3  *
   4  * The contents of this file are subject to the terms of the
   5  * Common Development and Distribution License (the "License").
   6  * You may not use this file except in compliance with the License.
   7  *
   8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
   9  * or http://www.opensolaris.org/os/licensing.
  10  * See the License for the specific language governing permissions
  11  * and limitations under the License.
  12  *
  13  * When distributing Covered Code, include this CDDL HEADER in each
  14  * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  15  * If applicable, add the following below this CDDL HEADER, with the
  16  * fields enclosed by brackets "[]" replaced with your own identifying
  17  * information: Portions Copyright [yyyy] [name of copyright owner]
  18  *
  19  * CDDL HEADER END
  20  */

  22 /*
  23  * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
  24  */

  26 #endif /* ! codereview */
  27 /*
  28  * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
  29  * Use is subject to license terms.
  30  */

  32 #if defined(__sparc)
  33 #include <stdio.h>
  34 #include <unistd.h>
  35 #include <string.h>
  36 #include <signal.h>
  37 #include <siginfo.h>
  38 #include <thread.h>
  39 #include <ucontext.h>
  40 #include <math.h>
  41 #if defined(__SUNPRO_C)
  42 #include <sunmath.h>
  43 #endif
  44 #include <fenv.h>

  46 #include "fenv_inlines.h"
  47 #include "libm_inlines.h"

  49 #ifdef __sparcv9

  50 #define FPreg(X)        &uap->uc_mcontext.fpregs.fpu_fr.fpu_regs[X]

  52 #define FPREG(X)        &uap->uc_mcontext.fpregs.fpu_fr.fpu_dregs[(X >> 1) | \
  53         ((X & 1) << 4)]
  28 #define FPREG(X)        &uap->uc_mcontext.fpregs.fpu_fr.fpu_dregs[(X>>1)| \
  29                                         ((X&1)<<4)]

  54 #else

  55 #include <sys/procfs.h>
```



```
  57 #define FPxreg(X)               & \
  58         ((prxregset_t *)uap->uc_mcontext.xrs.xrs_ptr)->pr_un.pr_v8p.pr_xfr. \
  59             pr_regs[X]
  35 #define FPxreg(X)       &((prxregset_t*)uap->uc_mcontext.xrs.xrs_ptr)->pr_un.pr_

  61 #define FPreg(X)                & uap->uc_mcontext.fpregs.fpu_fr.fpu_regs[X]
  37 #define FPreg(X)        &uap->uc_mcontext.fpregs.fpu_fr.fpu_regs[X]

  39 #define FPREG(X)        ((X & 1)? FPxreg(X - 1) : FPreg(X))

  63 #define FPREG(X)                ((X & 1) ? FPxreg(X - 1) : FPreg(X))
  64 #endif /* ! codereview */
  65 #endif /* __sparcv9 */

  67 #include "fex_handler.h"

  69 /* avoid dependence on libsunmath */
  70 static enum fp_class_type
  71 my_fp_classl(long double *a)
  72 {
  73         int msw = *(int *)a & ~0x80000000;
  41         int             msw = *(int*)a & ~0x80000000;

  75         if (msw >= 0x7fff0000) {
  76                 if (((msw & 0xffff) | *(1 + (int *)a) | *(2 + (int *)a) | *(3 +
  77                     (int *)a)) == 0)
  78                         return (fp_infinity);
  44                 if (((msw & 0xffff) | *(1+(int*)a) | *(2+(int*)a) | *(3+(int*)a)
  45                         return fp_infinity;
  79                 else if (msw & 0x8000)
  80                         return (fp_quiet);
  47                         return fp_quiet;
  81                 else
  82                         return (fp_signaling);
  49                         return fp_signaling;
  83         } else if (msw < 0x10000) {
  84                 if ((msw | *(1 + (int *)a) | *(2 + (int *)a) | *(3 +
  85                     (int *)a)) == 0)
  86                         return (fp_zero);
  51                 if ((msw | *(1+(int*)a) | *(2+(int*)a) | *(3+(int*)a)) == 0)
  52                         return fp_zero;
  87                 else
  88                         return (fp_subnormal);
  89         } else {
  90                 return (fp_normal);
  91         }
  54                         return fp_subnormal;
  55         } else
  56                 return fp_normal;
  92 }

  94 /*
  95  *  Determine which type of invalid operation exception occurred
  96  */
  60 *  Determine which type of invalid operation exception occurred
  61 */
  97 enum fex_exception
  98 __fex_get_invalid_type(siginfo_t *sip, ucontext_t *uap)
  99 {
 100         unsigned instr, opf, rs1, rs2;
 101         enum fp_class_type t1, t2;

 103         /* parse the instruction which caused the exception */
 104         instr = uap->uc_mcontext.fpregs.fpu_q->FQu.fpq.fpq_instr;
 105         opf = (instr >> 5) & 0x1ff;
 106         rs1 = (instr >> 14) & 0x1f;
```

```
107         rs2 = instr & 0x1f;

109         /* determine the classes of the operands */
110         switch (opf & 3) {
111         case 1:                         /* single */
112                 t1 = fp_classf(*(float *)FPreg(rs1));
113                 t2 = fp_classf(*(float *)FPreg(rs2));
 77                 t1 = fp_classf(*(float*)FPreg(rs1));
 78                 t2 = fp_classf(*(float*)FPreg(rs2));
114                 break;

116         case 2:                         /* double */
117                 t1 = fp_class(*(double *)FPREG(rs1));
118                 t2 = fp_class(*(double *)FPREG(rs2));
 82                 t1 = fp_class(*(double*)FPREG(rs1));
 83                 t2 = fp_class(*(double*)FPREG(rs2));
119                 break;

121         case 3:                         /* quad */
122                 t1 = my_fp_classl((long double *)FPREG(rs1));
123                 t2 = my_fp_classl((long double *)FPREG(rs2));
 87                 t1 = my_fp_classl((long double*)FPREG(rs1));
 88                 t2 = my_fp_classl((long double*)FPREG(rs2));
124                 break;

126         default:        /* integer operands never cause an invalid operation */
127                 return ((enum fex_exception)-1);
 92                 return (enum fex_exception) -1;
128         }

130         /* if rs2 is snan, return immediately */
131         if (t2 == fp_signaling)
132                 return (fex_inv_snan);
 97                 return fex_inv_snan;

134         /* determine the type of operation */
135         switch ((instr >> 19) & 0x183f) {
136         case 0x1034: /* add, subtract, multiply, divide, square root, convert */

138 #endif /* ! codereview */
139                 switch (opf & 0x1fc) {
140                 case 0x40:
141                 case 0x44:              /* add or subtract */

143 #endif /* ! codereview */
144                         if (t1 == fp_signaling)
145                                 return (fex_inv_snan);
102                                 return fex_inv_snan;
146                         else
147                                 return (fex_inv_isi);
104                                 return fex_inv_isi;

149                 case 0x48:
150                 case 0x68:
151                 case 0x6c:              /* multiply */

153 #endif /* ! codereview */
154                         if (t1 == fp_signaling)
155                                 return (fex_inv_snan);
109                                 return fex_inv_snan;
156                         else
157                                 return (fex_inv_zmi);
111                                 return fex_inv_zmi;

159                 case 0x4c:              /* divide */
```



```
161 #endif /* ! codereview */
162                         if (t1 == fp_signaling)
163                                 return (fex_inv_snan);
114                                 return fex_inv_snan;
164                         else if (t1 == fp_zero)
165                                 return (fex_inv_zdz);
116                                 return fex_inv_zdz;
166                         else
167                                 return (fex_inv_idi);
118                                 return fex_inv_idi;

169                 case 0x28:              /* square root */
170                         return (fex_inv_sqrt);
121                         return fex_inv_sqrt;

172                 case 0x80:
173                 case 0xd0:              /* convert to integer */
174                         return (fex_inv_int);
125                         return fex_inv_int;
175                 }

177 #endif /* ! codereview */
178                 break;

180         case 0x1035:                    /* compare */

182 #endif /* ! codereview */
183                 if (t1 == fp_signaling)
184                         return (fex_inv_snan);
127                         return fex_inv_snan;
185                 else
186                         return (fex_inv_cmp);
129                         return fex_inv_cmp;
187         }

189         return ((enum fex_exception)-1);
132         return (enum fex_exception) -1;
190 }

192 #ifdef __sparcv9
193 extern void _Qp_sqrt(long double *, const long double *);
194 #else
195 extern long double _Q_sqrt(long double);
196 #endif

198 /*
199  *  Get the operands, generate the default untrapped result with
200  *  exceptions, and set a code indicating the type of operation
201  */
142 *  Get the operands, generate the default untrapped result with
143 *  exceptions, and set a code indicating the type of operation
144 */
202 void
203 __fex_get_op(siginfo_t *sip, ucontext_t *uap, fex_info_t *info)
204 {
205         unsigned long fsr;
206         unsigned instr, opf, rs1, rs2;
207         volatile int c;

209         /* parse the instruction which caused the exception */
210         instr = uap->uc_mcontext.fpregs.fpu_q->FQu.fpq.fpq_instr;
211         opf = (instr >> 5) & 0x1ff;
212         rs1 = (instr >> 14) & 0x1f;
213         rs2 = instr & 0x1f;

215         /* get the operands */
```

```
        switch (opf & 3) {
        case 0:                         /* integer */
                info->op1.type = fex_nodata;

#endif /* ! codereview */
                if (opf & 0x40) {
                        info->op2.type = fex_int;
                        info->op2.val.i = *(int *)FPreg(rs2);
                } else {
                        info->op2.val.i = *(int*)FPreg(rs2);
                }
                else {
                        info->op2.type = fex_llong;
                        info->op2.val.l = *(long long *)FPREG(rs2);
                        info->op2.val.l = *(long long*)FPREG(rs2);
                }

#endif /* ! codereview */
                break;

        case 1:                         /* single */
                info->op1.type = info->op2.type = fex_float;
                info->op1.val.f = *(float *)FPreg(rs1);
                info->op2.val.f = *(float *)FPreg(rs2);
                info->op1.val.f = *(float*)FPreg(rs1);
                info->op2.val.f = *(float*)FPreg(rs2);
                break;

        case 2:                         /* double */
                info->op1.type = info->op2.type = fex_double;
                info->op1.val.d = *(double *)FPREG(rs1);
                info->op2.val.d = *(double *)FPREG(rs2);
                info->op1.val.d = *(double*)FPREG(rs1);
                info->op2.val.d = *(double*)FPREG(rs2);
                break;

        case 3:                         /* quad */
                info->op1.type = info->op2.type = fex_ldouble;
                info->op1.val.q = *(long double *)FPREG(rs1);
                info->op2.val.q = *(long double *)FPREG(rs2);
                info->op1.val.q = *(long double*)FPREG(rs1);
                info->op2.val.q = *(long double*)FPREG(rs2);
                break;
        }

        /*
         * initialize res to the default untrapped result and ex to the
         * corresponding flags (assume trapping is disabled and flags are
         * clear)
         */
        /* initialize res to the default untrapped result and ex to the
           corresponding flags (assume trapping is disabled and flags
           are clear) */
        info->op = fex_other;
        info->res.type = fex_nodata;

#endif /* ! codereview */
        switch ((instr >> 19) & 0x183f) {
        case 0x1035:                    /* compare */
                info->op = fex_cmp;

#endif /* ! codereview */
                switch (opf) {
                case 0x51:              /* compare single */
                        c = (info->op1.val.f == info->op2.val.f);
                        break;
```



```
                case 0x52:              /* compare double */
                        c = (info->op1.val.d == info->op2.val.d);
                        break;

                case 0x53:              /* compare quad */
                        c = (info->op1.val.q == info->op2.val.q);
                        break;

                case 0x55:              /* compare single with exception */
                        c = (info->op1.val.f < info->op2.val.f);
                        break;

                case 0x56:              /* compare double with exception */
                        c = (info->op1.val.d < info->op2.val.d);
                        break;

                case 0x57:              /* compare quad with exception */
                        c = (info->op1.val.q < info->op2.val.q);
                        break;
                }

#endif /* ! codereview */
                break;

        case 0x1034: /* add, subtract, multiply, divide, square root, convert */

#endif /* ! codereview */
                switch (opf) {
                case 0x41:              /* add single */
                        info->op = fex_add;
                        info->res.type = fex_float;
                        info->res.val.f = info->op1.val.f + info->op2.val.f;
                        break;

                case 0x42:              /* add double */
                        info->op = fex_add;
                        info->res.type = fex_double;
                        info->res.val.d = info->op1.val.d + info->op2.val.d;
                        break;

                case 0x43:              /* add quad */
                        info->op = fex_add;
                        info->res.type = fex_ldouble;
                        info->res.val.q = info->op1.val.q + info->op2.val.q;
                        break;

                case 0x45:              /* subtract single */
                        info->op = fex_sub;
                        info->res.type = fex_float;
                        info->res.val.f = info->op1.val.f - info->op2.val.f;
                        break;

                case 0x46:              /* subtract double */
                        info->op = fex_sub;
                        info->res.type = fex_double;
                        info->res.val.d = info->op1.val.d - info->op2.val.d;
                        break;

                case 0x47:              /* subtract quad */
                        info->op = fex_sub;
                        info->res.type = fex_ldouble;
                        info->res.val.q = info->op1.val.q - info->op2.val.q;
                        break;

                case 0x49:              /* multiply single */
```

```
                        info->op = fex_mul;
                        info->res.type = fex_float;
                        info->res.val.f = info->op1.val.f * info->op2.val.f;
                        break;

                case 0x4a:              /* multiply double */
                        info->op = fex_mul;
                        info->res.type = fex_double;
                        info->res.val.d = info->op1.val.d * info->op2.val.d;
                        break;

                case 0x4b:              /* multiply quad */
                        info->op = fex_mul;
                        info->res.type = fex_ldouble;
                        info->res.val.q = info->op1.val.q * info->op2.val.q;
                        break;

                case 0x69:              /* fsmuld */
                        info->op = fex_mul;
                        info->res.type = fex_double;
                        info->res.val.d = (double)info->op1.val.f *
                            (double)info->op2.val.f;
                        info->res.val.d = (double)info->op1.val.f * (double)info
                        break;

                case 0x6e:              /* fdmulq */
                        info->op = fex_mul;
                        info->res.type = fex_ldouble;
                        info->res.val.q = (long double)info->op1.val.d * (long
                            double)info->op2.val.d;
                        info->res.val.q = (long double)info->op1.val.d *
                                (long double)info->op2.val.d;
                        break;

                case 0x4d:              /* divide single */
                        info->op = fex_div;
                        info->res.type = fex_float;
                        info->res.val.f = info->op1.val.f / info->op2.val.f;
                        break;

                case 0x4e:              /* divide double */
                        info->op = fex_div;
                        info->res.type = fex_double;
                        info->res.val.d = info->op1.val.d / info->op2.val.d;
                        break;

                case 0x4f:              /* divide quad */
                        info->op = fex_div;
                        info->res.type = fex_ldouble;
                        info->res.val.q = info->op1.val.q / info->op2.val.q;
                        break;

                case 0x29:              /* square root single */
                        info->op = fex_sqrt;
                        info->op1 = info->op2;
                        info->op2.type = fex_nodata;
                        info->res.type = fex_float;
                        info->res.val.f = sqrtf(info->op1.val.f);
                        break;

                case 0x2a:              /* square root double */
                        info->op = fex_sqrt;
                        info->op1 = info->op2;
                        info->op2.type = fex_nodata;
                        info->res.type = fex_double;
                        info->res.val.d = sqrt(info->op1.val.d);
```



```
                        break;

                case 0x2b:              /* square root quad */
                        info->op = fex_sqrt;
                        info->op1 = info->op2;
                        info->op2.type = fex_nodata;
                        info->res.type = fex_ldouble;
#ifdef __sparcv9
                        _Qp_sqrt(&info->res.val.q, &info->op1.val.q);
#else
                        info->res.val.q = _Q_sqrt(info->op1.val.q);
#endif
                        break;

                default:                /* conversions */
                        info->op = fex_cnvt;
                        info->op1 = info->op2;
                        info->op2.type = fex_nodata;

#endif /* ! codereview */
                        switch (opf) {
                        case 0xd1:      /* convert single to int */
                                info->res.type = fex_int;
                                info->res.val.i = (int)info->op1.val.f;
                                info->res.val.i = (int) info->op1.val.f;
                                break;

                        case 0xd2:      /* convert double to int */
                                info->res.type = fex_int;
                                info->res.val.i = (int)info->op1.val.d;
                                info->res.val.i = (int) info->op1.val.d;
                                break;

                        case 0xd3:      /* convert quad to int */
                                info->res.type = fex_int;
                                info->res.val.i = (int)info->op1.val.q;
                                info->res.val.i = (int) info->op1.val.q;
                                break;

                        case 0x81:      /* convert single to long long */
                                info->res.type = fex_llong;
                                info->res.val.l = (long long)info->op1.val.f;
                                info->res.val.l = (long long) info->op1.val.f;
                                break;

                        case 0x82:      /* convert double to long long */
                                info->res.type = fex_llong;
                                info->res.val.l = (long long)info->op1.val.d;
                                info->res.val.l = (long long) info->op1.val.d;
                                break;

                        case 0x83:      /* convert quad to long long */
                                info->res.type = fex_llong;
                                info->res.val.l = (long long)info->op1.val.q;
                                info->res.val.l = (long long) info->op1.val.q;
                                break;

                        case 0xc4:      /* convert int to single */
                                info->res.type = fex_float;
                                info->res.val.f = (float)info->op1.val.i;
                                info->res.val.f = (float) info->op1.val.i;
                                break;

                        case 0x84:      /* convert long long to single */
                                info->res.type = fex_float;
                                info->res.val.f = (float)info->op1.val.l;
```

```
285                                 info->res.val.f = (float) info->op1.val.l;
457                                 break;

459                         case 0x88:      /* convert long long to double */
460                                 info->res.type = fex_double;
461                                 info->res.val.d = (double)info->op1.val.l;
290                                 info->res.val.d = (double) info->op1.val.l;
462                                 break;

464                         case 0xc6:      /* convert double to single */
465                                 info->res.type = fex_float;
466                                 info->res.val.f = (float)info->op1.val.d;
295                                 info->res.val.f = (float) info->op1.val.d;
467                                 break;

469                         case 0xc7:      /* convert quad to single */
470                                 info->res.type = fex_float;
471                                 info->res.val.f = (float)info->op1.val.q;
300                                 info->res.val.f = (float) info->op1.val.q;
472                                 break;

474                         case 0xc9:      /* convert single to double */
475                                 info->res.type = fex_double;
476                                 info->res.val.d = (double)info->op1.val.f;
305                                 info->res.val.d = (double) info->op1.val.f;
477                                 break;

479                         case 0xcb:      /* convert quad to double */
480                                 info->res.type = fex_double;
481                                 info->res.val.d = (double)info->op1.val.q;
310                                 info->res.val.d = (double) info->op1.val.q;
482                                 break;

484                         case 0xcd:      /* convert single to quad */
485                                 info->res.type = fex_ldouble;
486                                 info->res.val.q = (long double)info->op1.val.f;
315                                 info->res.val.q = (long double) info->op1.val.f;
487                                 break;

489                         case 0xce:      /* convert double to quad */
490                                 info->res.type = fex_ldouble;
491                                 info->res.val.q = (long double)info->op1.val.d;
320                                 info->res.val.q = (long double) info->op1.val.d;
492                                 break;
493                         }
494                 }

496 #endif /* ! codereview */
497                 break;
498         }

500 #endif /* ! codereview */
501         __fenv_getfsr(&fsr);
502         info->flags = (int)__fenv_get_ex(fsr);
503         __fenv_set_ex(fsr, 0);
504         __fenv_setfsr(&fsr);
505 }

507 /*
508  *  Store the specified result; if no result is given but the exception
509  *  is underflow or overflow, supply the default trapped result
510  */
324 *  Store the specified result; if no result is given but the exception
325 *  is underflow or overflow, supply the default trapped result
326 */
511 void
```



```
512 __fex_st_result(siginfo_t *sip, ucontext_t *uap, fex_info_t *info)
513 {
514         unsigned instr, opf, rs1, rs2, rd;
515         long double qscl;
516         double dscl;
517         float fscl;

519         /* parse the instruction which caused the exception */
520         instr = uap->uc_mcontext.fpregs.fpu_q->FQu.fpq.fpq_instr;
521         opf = (instr >> 5) & 0x1ff;
522         rs1 = (instr >> 14) & 0x1f;
523         rs2 = instr & 0x1f;
524         rd = (instr >> 25) & 0x1f;

526         /* if the instruction is a compare, just set fcc to unordered */
527         if (((instr >> 19) & 0x183f) == 0x1035) {
528                 if (rd == 0) {
344                 if (rd == 0)
529                         uap->uc_mcontext.fpregs.fpu_fsr |= 0xc00;
530                 } else {
346                 else {
531 #ifdef __sparcv9
532                         uap->uc_mcontext.fpregs.fpu_fsr |= (3l << ((rd << 1) +
533                             30));
348                         uap->uc_mcontext.fpregs.fpu_fsr |= (3l << ((rd << 1) + 3
534 #else
535                         ((prxregset_t *)uap->uc_mcontext.xrs.xrs_ptr)->pr_un.
536                             pr_v8p.pr_xfsr |= (3 << ((rd - 1) << 1));
350                         ((prxregset_t*)uap->uc_mcontext.xrs.xrs_ptr)->pr_un.pr_v
537 #endif
538                 }

540 #endif /* ! codereview */
541                 return;
542         }

544         /*
545          * if there is no result available, try to generate the untrapped
546          * default
547          */
353         /* if there is no result available, try to generate the untrapped
354            default */
548         if (info->res.type == fex_nodata) {
549                 /* set scale factors for exponent wrapping */
550                 switch (sip->si_code) {
551                 case FPE_FLTOVF:
552                         /* 2^-96 */
553                         fscl = 1.262177448e-29f;
554                         /* 2^-768 */
555                         dscl = 6.441148769597133308e-232;
556                         /* 2^-12288 */
557                         qscl = 8.778357852076208839765066529179033145e-3700l;
359                         fscl = 1.262177448e-29f;        /* 2^-96 */
360                         dscl = 6.441148769597133308e-232;       /* 2^-768 */
361                         qscl = 8.778357852076208839765066529179033145e-3700l;/*
558                         break;

560                 case FPE_FLTUND:
561                         /* 2^96 */
562                         fscl = 7.922816251e+28f;
563                         /* 2^768 */
564                         dscl = 1.552518092300708935e+231;
565                         /* 2^12288 */
566                         qscl = 1.139165225263043370845938579315932009e+3699l;
365                         fscl = 7.922816251e+28f;        /* 2^96 */
366                         dscl = 1.552518092300708935e+231;       /* 2^768 */
```

```
                        qscl = 1.139165225263043370845938579315932009e+3699l;/*
                        break;

                default:

                        /*
                         * user may have blown away the default result by
                         * mistake, so try to regenerate it
                         */
                        /* user may have blown away the default result by mistak
                           so try to regenerate it */
                        (void) __fex_get_op(sip, uap, info);

#endif /* ! codereview */
                        if (info->res.type != fex_nodata)
                                goto stuff;

#endif /* ! codereview */
                        /* couldn't do it */
                        return;
                }

                /* get the operands */
                switch (opf & 3) {
                case 1:                 /* single */
                        info->op1.val.f = *(float *)FPreg(rs1);
                        info->op2.val.f = *(float *)FPreg(rs2);
                        info->op1.val.f = *(float*)FPreg(rs1);
                        info->op2.val.f = *(float*)FPreg(rs2);
                        break;

                case 2:                 /* double */
                        info->op1.val.d = *(double *)FPREG(rs1);
                        info->op2.val.d = *(double *)FPREG(rs2);
                        info->op1.val.d = *(double*)FPREG(rs1);
                        info->op2.val.d = *(double*)FPREG(rs2);
                        break;

                case 3:                 /* quad */
                        info->op1.val.q = *(long double *)FPREG(rs1);
                        info->op2.val.q = *(long double *)FPREG(rs2);
                        info->op1.val.q = *(long double*)FPREG(rs1);
                        info->op2.val.q = *(long double*)FPREG(rs2);
                        break;
                }

                /* generate the wrapped result */
                switch (opf) {
                case 0x41:              /* add single */
                        info->res.type = fex_float;
                        info->res.val.f = fscl * (fscl * info->op1.val.f +
                            fscl * info->op2.val.f);
                        break;

                case 0x42:              /* add double */
                        info->res.type = fex_double;
                        info->res.val.d = dscl * (dscl * info->op1.val.d +
                            dscl * info->op2.val.d);
                        break;

                case 0x43:              /* add quad */
                        info->res.type = fex_ldouble;
                        info->res.val.q = qscl * (qscl * info->op1.val.q +
                            qscl * info->op2.val.q);
                        break;
```



```
                case 0x45:              /* subtract single */
                        info->res.type = fex_float;
                        info->res.val.f = fscl * (fscl * info->op1.val.f -
                            fscl * info->op2.val.f);
                        break;

                case 0x46:              /* subtract double */
                        info->res.type = fex_double;
                        info->res.val.d = dscl * (dscl * info->op1.val.d -
                            dscl * info->op2.val.d);
                        break;

                case 0x47:              /* subtract quad */
                        info->res.type = fex_ldouble;
                        info->res.val.q = qscl * (qscl * info->op1.val.q -
                            qscl * info->op2.val.q);
                        break;

                case 0x49:              /* multiply single */
                        info->res.type = fex_float;
                        info->res.val.f = (fscl * info->op1.val.f) * (fscl *
                            info->op2.val.f);
                        info->res.val.f = (fscl * info->op1.val.f) *
                                (fscl * info->op2.val.f);
                        break;

                case 0x4a:              /* multiply double */
                        info->res.type = fex_double;
                        info->res.val.d = (dscl * info->op1.val.d) * (dscl *
                            info->op2.val.d);
                        info->res.val.d = (dscl * info->op1.val.d) *
                                (dscl * info->op2.val.d);
                        break;

                case 0x4b:              /* multiply quad */
                        info->res.type = fex_ldouble;
                        info->res.val.q = (qscl * info->op1.val.q) * (qscl *
                            info->op2.val.q);
                        info->res.val.q = (qscl * info->op1.val.q) *
                                (qscl * info->op2.val.q);
                        break;

                case 0x4d:              /* divide single */
                        info->res.type = fex_float;
                        info->res.val.f = (fscl * info->op1.val.f) /
                            (info->op2.val.f / fscl);
                        break;

                case 0x4e:              /* divide double */
                        info->res.type = fex_double;
                        info->res.val.d = (dscl * info->op1.val.d) /
                            (info->op2.val.d / dscl);
                        break;

                case 0x4f:              /* divide quad */
                        info->res.type = fex_ldouble;
                        info->res.val.q = (qscl * info->op1.val.q) /
                            (info->op2.val.q / qscl);
                        break;

                case 0xc6:              /* convert double to single */
                        info->res.type = fex_float;
                        info->res.val.f = (float)(fscl * (fscl *
                            info->op1.val.d));
                        info->res.val.f = (float) (fscl * (fscl * info->op1.val.
                        break;
```

```
684                 case 0xc7:              /* convert quad to single */
685                         info->res.type = fex_float;
686                         info->res.val.f = (float)(fscl * (fscl *
687                             info->op1.val.q));
470                         info->res.val.f = (float) (fscl * (fscl * info->op1.val.
688                         break;

690                 case 0xcb:              /* convert quad to double */
691                         info->res.type = fex_double;
692                         info->res.val.d = (double)(dscl * (dscl *
693                             info->op1.val.q));
475                         info->res.val.d = (double) (dscl * (dscl * info->op1.val
694                         break;
695                 }

697                 if (info->res.type == fex_nodata)
698                         /* couldn't do it */
699                         return;
700         }

702 stuff:
703         /* stick the result in the destination */
704         if (opf & 0x80) {               /* conversion */
705                 if (opf & 0x10) {       /* result is an int */
706                         switch (info->res.type) {
707                         case fex_llong:
708                                 info->res.val.i = (int)info->res.val.l;
490                                 info->res.val.i = (int) info->res.val.l;
709                                 break;

711                         case fex_float:
712                                 info->res.val.i = (int)info->res.val.f;
494                                 info->res.val.i = (int) info->res.val.f;
713                                 break;

715                         case fex_double:
716                                 info->res.val.i = (int)info->res.val.d;
498                                 info->res.val.i = (int) info->res.val.d;
717                                 break;

719                         case fex_ldouble:
720                                 info->res.val.i = (int)info->res.val.q;
502                                 info->res.val.i = (int) info->res.val.q;
721                                 break;

723                         default:
724                                 break;
725                         }

727                         *(int *)FPreg(rd) = info->res.val.i;
508                         *(int*)FPreg(rd) = info->res.val.i;
728                         return;
729                 }

731                 switch (opf & 0xc) {
732                 case 0:                 /* result is long long */

734 #endif /* ! codereview */
735                         switch (info->res.type) {
736                         case fex_int:
737                                 info->res.val.l = (long long)info->res.val.i;
514                                 info->res.val.l = (long long) info->res.val.i;
738                                 break;

740                         case fex_float:
```



```
741                                 info->res.val.l = (long long)info->res.val.f;
518                                 info->res.val.l = (long long) info->res.val.f;
742                                 break;

744                         case fex_double:
745                                 info->res.val.l = (long long)info->res.val.d;
522                                 info->res.val.l = (long long) info->res.val.d;
746                                 break;

748                         case fex_ldouble:
749                                 info->res.val.l = (long long)info->res.val.q;
526                                 info->res.val.l = (long long) info->res.val.q;
750                                 break;

752                         default:
753                                 break;
754                         }

756                         *(long long *)FPREG(rd) = info->res.val.l;
532                         *(long long*)FPREG(rd) = info->res.val.l;
757                         break;

759                 case 0x4:               /* result is float */

761 #endif /* ! codereview */
762                         switch (info->res.type) {
763                         case fex_int:
764                                 info->res.val.f = (float)info->res.val.i;
536                                 info->res.val.f = (float) info->res.val.i;
765                                 break;

767                         case fex_llong:
768                                 info->res.val.f = (float)info->res.val.l;
540                                 info->res.val.f = (float) info->res.val.l;
769                                 break;

771                         case fex_double:
772                                 info->res.val.f = (float)info->res.val.d;
544                                 info->res.val.f = (float) info->res.val.d;
773                                 break;

775                         case fex_ldouble:
776                                 info->res.val.f = (float)info->res.val.q;
548                                 info->res.val.f = (float) info->res.val.q;
777                                 break;

779                         default:
780                                 break;
781                         }

783                         *(float *)FPreg(rd) = info->res.val.f;
554                         *(float*)FPreg(rd) = info->res.val.f;
784                         break;

786                 case 0x8:               /* result is double */

788 #endif /* ! codereview */
789                         switch (info->res.type) {
790                         case fex_int:
791                                 info->res.val.d = (double)info->res.val.i;
558                                 info->res.val.d = (double) info->res.val.i;
792                                 break;

794                         case fex_llong:
795                                 info->res.val.d = (double)info->res.val.l;
562                                 info->res.val.d = (double) info->res.val.l;
```

```
796                                 break;

798                         case fex_float:
799                                 info->res.val.d = (double)info->res.val.f;
566                                 info->res.val.d = (double) info->res.val.f;
800                                 break;

802                         case fex_ldouble:
803                                 info->res.val.d = (double)info->res.val.q;
570                                 info->res.val.d = (double) info->res.val.q;
804                                 break;

806                         default:
807                                 break;
808                         }

810                         *(double *)FPREG(rd) = info->res.val.d;
576                         *(double*)FPREG(rd) = info->res.val.d;
811                         break;

813                 case 0xc:               /* result is long double */

815 #endif /* ! codereview */
816                         switch (info->res.type) {
817                         case fex_int:
818                                 info->res.val.q = (long double)info->res.val.i;
580                                 info->res.val.q = (long double) info->res.val.i;
819                                 break;

821                         case fex_llong:
822                                 info->res.val.q = (long double)info->res.val.l;
584                                 info->res.val.q = (long double) info->res.val.l;
823                                 break;

825                         case fex_float:
826                                 info->res.val.q = (long double)info->res.val.f;
588                                 info->res.val.q = (long double) info->res.val.f;
827                                 break;

829                         case fex_double:
830                                 info->res.val.q = (long double)info->res.val.d;
592                                 info->res.val.q = (long double) info->res.val.d;
831                                 break;

833                         default:
834                                 break;
835                         }

837                         *(long double *)FPREG(rd) = info->res.val.q;
598                         *(long double*)FPREG(rd) = info->res.val.q;
838                         break;
839                 }

841 #endif /* ! codereview */
842                 return;
843         }

845         if ((opf & 0xf0) == 0x60) {     /* fsmuld, fdmulq */
846                 switch (opf & 0xc0) {
847                 case 0x8:               /* result is double */

849 #endif /* ! codereview */
850                         switch (info->res.type) {
851                         case fex_int:
852                                 info->res.val.d = (double)info->res.val.i;
601                                 info->res.val.d = (double) info->res.val.i;
```



```
853                                 break;

855                         case fex_llong:
856                                 info->res.val.d = (double)info->res.val.l;
605                                 info->res.val.d = (double) info->res.val.l;
857                                 break;

859                         case fex_float:
860                                 info->res.val.d = (double)info->res.val.f;
609                                 info->res.val.d = (double) info->res.val.f;
861                                 break;

863                         case fex_ldouble:
864                                 info->res.val.d = (double)info->res.val.q;
613                                 info->res.val.d = (double) info->res.val.q;
865                                 break;

867                         default:
868                                 break;
869                         }

871                         *(double *)FPREG(rd) = info->res.val.d;
619                         *(double*)FPREG(rd) = info->res.val.d;
872                         break;

874                 case 0xc:               /* result is long double */

876 #endif /* ! codereview */
877                         switch (info->res.type) {
878                         case fex_int:
879                                 info->res.val.q = (long double)info->res.val.i;
623                                 info->res.val.q = (long double) info->res.val.i;
880                                 break;

882                         case fex_llong:
883                                 info->res.val.q = (long double)info->res.val.l;
627                                 info->res.val.q = (long double) info->res.val.l;
884                                 break;

886                         case fex_float:
887                                 info->res.val.q = (long double)info->res.val.f;
631                                 info->res.val.q = (long double) info->res.val.f;
888                                 break;

890                         case fex_double:
891                                 info->res.val.q = (long double)info->res.val.d;
635                                 info->res.val.q = (long double) info->res.val.d;
892                                 break;

894                         default:
895                                 break;
896                         }

898                         *(long double *)FPREG(rd) = info->res.val.q;
641                         *(long double*)FPREG(rd) = info->res.val.q;
899                         break;
900                 }

902 #endif /* ! codereview */
903                 return;
904         }

906         switch (opf & 3) {              /* other arithmetic op */
907         case 1:                         /* result is float */

909 #endif /* ! codereview */
```

```
910                 switch (info->res.type) {
911                 case fex_int:
912                         info->res.val.f = (float)info->res.val.i;
644                         info->res.val.f = (float) info->res.val.i;
913                         break;

915                 case fex_llong:
916                         info->res.val.f = (float)info->res.val.l;
648                         info->res.val.f = (float) info->res.val.l;
917                         break;

919                 case fex_double:
920                         info->res.val.f = (float)info->res.val.d;
652                         info->res.val.f = (float) info->res.val.d;
921                         break;

923                 case fex_ldouble:
924                         info->res.val.f = (float)info->res.val.q;
656                         info->res.val.f = (float) info->res.val.q;
925                         break;

927                 default:
928                         break;
929                 }

931                 *(float *)FPreg(rd) = info->res.val.f;
662                 *(float*)FPreg(rd) = info->res.val.f;
932                 break;

934         case 2:                         /* result is double */

936 #endif /* ! codereview */
937                 switch (info->res.type) {
938                 case fex_int:
939                         info->res.val.d = (double)info->res.val.i;
666                         info->res.val.d = (double) info->res.val.i;
940                         break;

942                 case fex_llong:
943                         info->res.val.d = (double)info->res.val.l;
670                         info->res.val.d = (double) info->res.val.l;
944                         break;

946                 case fex_float:
947                         info->res.val.d = (double)info->res.val.f;
674                         info->res.val.d = (double) info->res.val.f;
948                         break;

950                 case fex_ldouble:
951                         info->res.val.d = (double)info->res.val.q;
678                         info->res.val.d = (double) info->res.val.q;
952                         break;

954                 default:
955                         break;
956                 }

958                 *(double *)FPREG(rd) = info->res.val.d;
684                 *(double*)FPREG(rd) = info->res.val.d;
959                 break;

961         case 3:                         /* result is long double */

963 #endif /* ! codereview */
964                 switch (info->res.type) {
965                 case fex_int:
```



```
966                         info->res.val.q = (long double)info->res.val.i;
688                         info->res.val.q = (long double) info->res.val.i;
967                         break;

969                 case fex_llong:
970                         info->res.val.q = (long double)info->res.val.l;
692                         info->res.val.q = (long double) info->res.val.l;
971                         break;

973                 case fex_float:
974                         info->res.val.q = (long double)info->res.val.f;
696                         info->res.val.q = (long double) info->res.val.f;
975                         break;

977                 case fex_double:
978                         info->res.val.q = (long double)info->res.val.d;
700                         info->res.val.q = (long double) info->res.val.d;
979                         break;

981                 default:
982                         break;
983                 }

985                 *(long double *)FPREG(rd) = info->res.val.q;
706                 *(long double*)FPREG(rd) = info->res.val.q;
986                 break;
987         }
988 }
```

_____*unchanged_portion_omitted_*

```
**********************************************************
   39389 Thu Jun  6 20:40:06 2019
new/usr/src/lib/libm/common/m9x/__fex_sse.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#include <ucontext.h>
#include <fenv.h>
#if defined(__SUNPRO_C)
#include <sunmath.h>
#else
#include <sys/ieeefp.h>
#endif
#include "fex_handler.h"
#include "fenv_inlines.h"

#if !defined(REG_PC)
#define REG_PC          EIP
#endif

#if !defined(REG_PS)
#define REG_PS          EFL
#endif

#ifdef __amd64
#define regno(X)        ((X < 4) ? REG_RAX - X : ((X > 4) ? REG_RAX + 1 - X : \
        REG_RSP))
#define regno(X)        ((X < 4)? REG_RAX - X : \
                        ((X > 4)? REG_RAX + 1 - X : REG_RSP))
#else
#define regno(X)        (EAX - X)
#endif

/*
 * Support for SSE instructions
 */
```



```
/*
 * Decode an SSE instruction.  Fill in *inst and return the length of the
 * instruction in bytes.  Return 0 if the instruction is not recognized.
 */
int
__fex_parse_sse(ucontext_t *uap, sseinst_t *inst)
{
        unsigned char *ip;
        char *addr;
        int i, dbl, simd, rex, modrm, sib, r;

        i = 0;
        ip = (unsigned char *)uap->uc_mcontext.gregs[REG_PC];

        /* look for pseudo-prefixes */
        dbl = 0;
        simd = SIMD;

#endif /* ! codereview */
        if (ip[i] == 0xF3) {
                simd = 0;
                i++;
        } else if (ip[i] == 0x66) {
                dbl = DOUBLE;
                i++;
        } else if (ip[i] == 0xF2) {
                dbl = DOUBLE;
                simd = 0;
                i++;
        }

        /* look for AMD64 REX prefix */
        rex = 0;

#endif /* ! codereview */
        if (ip[i] >= 0x40 && ip[i] <= 0x4F) {
                rex = ip[i];
                i++;
        }

        /* parse opcode */
        if (ip[i++] != 0x0F)
                return (0);

                return 0;
        switch (ip[i++]) {
        case 0x2A:
                inst->op = (int)cvtsi2ss + simd + dbl;

#endif /* ! codereview */
                if (!simd)
                        inst->op = (int)inst->op + (rex & 8);

#endif /* ! codereview */
                break;

        case 0x2C:
                inst->op = (int)cvttss2si + simd + dbl;

#endif /* ! codereview */
                if (!simd)
                        inst->op = (int)inst->op + (rex & 8);

#endif /* ! codereview */
                break;
```

```
126         case 0x2D:
127                 inst->op = (int)cvtss2si + simd + dbl;

129 #endif /* ! codereview */
130                 if (!simd)
131                         inst->op = (int)inst->op + (rex & 8);

133 #endif /* ! codereview */
134                 break;

136         case 0x2E:

138 #endif /* ! codereview */
139                 /* oddball: scalar instruction in a SIMD opcode group */
140                 if (!simd)
141                         return (0);

 56                         return 0;
143                 inst->op = (int)ucomiss + dbl;
144                 break;

146         case 0x2F:

148 #endif /* ! codereview */
149                 /* oddball: scalar instruction in a SIMD opcode group */
150                 if (!simd)
151                         return (0);

 61                         return 0;
153                 inst->op = (int)comiss + dbl;
154                 break;

156         case 0x51:
157                 inst->op = (int)sqrtss + simd + dbl;
158                 break;

160         case 0x58:
161                 inst->op = (int)addss + simd + dbl;
162                 break;

164         case 0x59:
165                 inst->op = (int)mulss + simd + dbl;
166                 break;

168         case 0x5A:
169                 inst->op = (int)cvtss2sd + simd + dbl;
170                 break;

172         case 0x5B:

174 #endif /* ! codereview */
175                 if (dbl) {
176                         if (simd)
177                                 inst->op = cvtps2dq;
178                         else
179                                 return (0);
 82                                 return 0;
180                 } else {
181                         inst->op = (simd) ? cvtdq2ps : cvttps2dq;
 84                         inst->op = (simd)? cvtdq2ps : cvttps2dq;
182                 }

184 #endif /* ! codereview */
185                 break;
```



```
187         case 0x5C:
188                 inst->op = (int)subss + simd + dbl;
189                 break;

191         case 0x5D:
192                 inst->op = (int)minss + simd + dbl;
193                 break;

195         case 0x5E:
196                 inst->op = (int)divss + simd + dbl;
197                 break;

199         case 0x5F:
200                 inst->op = (int)maxss + simd + dbl;
201                 break;

203         case 0xC2:
204                 inst->op = (int)cmpss + simd + dbl;
205                 break;

207         case 0xE6:

209 #endif /* ! codereview */
210                 if (simd) {
211                         if (dbl)
212                                 inst->op = cvttpd2dq;
213                         else
214                                 return (0);
 86                                 return 0;
215                 } else {
216                         inst->op = (dbl) ? cvtpd2dq : cvtdq2pd;
 88                         inst->op = (dbl)? cvtpd2dq : cvtdq2pd;
217                 }

219 #endif /* ! codereview */
220                 break;

222         default:
223                 return (0);
 90                 return 0;
224         }

226         /* locate operands */
227         modrm = ip[i++];

229         if (inst->op == cvtss2si || inst->op == cvttss2si || inst->op ==
230             cvtsd2si || inst->op == cvttsd2si || inst->op == cvtss2siq ||
231             inst->op == cvttss2siq || inst->op == cvtsd2siq || inst->op ==
232             cvttsd2siq) {
 96         if (inst->op == cvtss2si || inst->op == cvttss2si ||
 97             inst->op == cvtsd2si || inst->op == cvttsd2si ||
 98             inst->op == cvtss2siq || inst->op == cvttss2siq ||
 99             inst->op == cvtsd2siq || inst->op == cvttsd2siq) {
233                 /* op1 is a gp register */
234                 r = ((rex & 4) << 1) | ((modrm >> 3) & 7);
235                 inst->op1 = (sseoperand_t *)&uap->uc_mcontext.gregs[regno(r)];
236         } else if (inst->op == cvtps2pi || inst->op == cvttps2pi || inst->op ==
237             cvtpd2pi || inst->op == cvttpd2pi) {
103         } else if (inst->op == cvtps2pi || inst->op == cvttps2pi ||
104             inst->op == cvtpd2pi || inst->op == cvttpd2pi) {
238                 /* op1 is a mmx register */
239 #ifdef __amd64
240                 inst->op1 = (sseoperand_t *)
241                     &uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state
242                     .st[(modrm >> 3) & 7];
107                 inst->op1 = (sseoperand_t *)&uap->uc_mcontext.fpregs.fp_reg_set.
```

```
                        fpchip_state.st[(modrm >> 3) & 7];
#else
                inst->op1 = (sseoperand_t *)(10 * ((modrm >> 3) & 7) +
                        (char *)&uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state
                        .state[7]);
                        (char *)&uap->uc_mcontext.fpregs.fp_reg_set.
                        fpchip_state.state[7]);
#endif
        } else {
                /* op1 is a xmm register */
                r = ((rex & 4) << 1) | ((modrm >> 3) & 7);
                inst->op1 =
                        (sseoperand_t *)&uap->uc_mcontext.fpregs.fp_reg_set
                        .fpchip_state.xmm[r];
                inst->op1 = (sseoperand_t *)&uap->uc_mcontext.fpregs.
                        fp_reg_set.fpchip_state.xmm[r];
        }

        if ((modrm >> 6) == 3) {
                if (inst->op == cvtsi2ss || inst->op == cvtsi2sd || inst->op ==
                        cvtsi2ssq || inst->op == cvtsi2sdq) {
                if (inst->op == cvtsi2ss || inst->op == cvtsi2sd ||
                        inst->op == cvtsi2ssq || inst->op == cvtsi2sdq) {
                                /* op2 is a gp register */
                                r = ((rex & 1) << 3) | (modrm & 7);
                                inst->op2 =
                                        (sseoperand_t *)&uap->uc_mcontext.gregs[regno(r)];
                                inst->op2 = (sseoperand_t *)&uap->uc_mcontext.
                                        gregs[regno(r)];
                } else if (inst->op == cvtpi2ps || inst->op == cvtpi2pd) {
                                /* op2 is a mmx register */
#ifdef __amd64
                                inst->op2 =
                                        (sseoperand_t *)&uap->uc_mcontext.fpregs.fp_reg_set
                                        .fpchip_state.st[modrm & 7];
                                inst->op2 = (sseoperand_t *)&uap->uc_mcontext.fpregs.
                                        fp_reg_set.fpchip_state.st[modrm & 7];
#else
                                inst->op2 = (sseoperand_t *)(10 * (modrm & 7) +
                                        (char *)&uap->uc_mcontext.fpregs.fp_reg_set
                                        .fpchip_state.state[7]);
                                        (char *)&uap->uc_mcontext.fpregs.fp_reg_set.
                                        fpchip_state.state[7]);
#endif
                } else {
                                /* op2 is a xmm register */
                                r = ((rex & 1) << 3) | (modrm & 7);
                                inst->op2 =
                                        (sseoperand_t *)&uap->uc_mcontext.fpregs.fp_reg_set
                                        .fpchip_state.xmm[r];
                                inst->op2 = (sseoperand_t *)&uap->uc_mcontext.fpregs.
                                        fp_reg_set.fpchip_state.xmm[r];
                }
        } else if ((modrm & 0xc7) == 0x05) {
#ifdef __amd64
                /* address of next instruction + offset */
                r = i + 4;

                if (inst->op == cmpss || inst->op == cmpps || inst->op ==
                        cmpsd || inst->op == cmppd)
                if (inst->op == cmpss || inst->op == cmpps ||
                        inst->op == cmpsd || inst->op == cmppd)
                                r++;

#endif /* ! codereview */
                inst->op2 = (sseoperand_t *)(ip + r + *(int *)(ip + i));
```



```
#else
                /* absolute address */
                inst->op2 = (sseoperand_t *)(*(int *)(ip + i));
#endif
                i += 4;
        } else {
                /* complex address */
                if ((modrm & 7) == 4) {
                        /* parse sib byte */
                        sib = ip[i++];

#endif /* ! codereview */
                        if ((sib & 7) == 5 && (modrm >> 6) == 0) {
                                /* start with absolute address */
                                addr = (char *)(uintptr_t)(*(int *)(ip + i));
                                i += 4;
                        } else {
                                /* start with base */
                                r = ((rex & 1) << 3) | (sib & 7);
                                addr = (char *)uap->uc_mcontext.gregs[regno(r)];
                        }

#endif /* ! codereview */
                        r = ((rex & 2) << 2) | ((sib >> 3) & 7);

#endif /* ! codereview */
                        if (r != 4) {
                                /* add scaled index */
                                addr += uap->uc_mcontext.gregs[regno(r)] <<
                                        (sib >> 6);
                                addr += uap->uc_mcontext.gregs[regno(r)]
                                        << (sib >> 6);
                        }
                } else {
                        r = ((rex & 1) << 3) | (modrm & 7);
                        addr = (char *)uap->uc_mcontext.gregs[regno(r)];
                }

                /* add displacement, if any */
                if ((modrm >> 6) == 1) {
                        addr += (char)ip[i++];
                } else if ((modrm >> 6) == 2) {
                        addr += *(int *)(ip + i);
                        i += 4;
                }

#endif /* ! codereview */
                inst->op2 = (sseoperand_t *)addr;
        }

        if (inst->op == cmpss || inst->op == cmpps || inst->op == cmpsd ||
            inst->op == cmppd) {
                /* get the immediate operand */
                inst->imm = ip[i++];
        }

        return (i);
        return i;
}

static enum fp_class_type
my_fp_classf(float *x)
{
        int i = *(int *)x & ~0x80000000;

        if (i < 0x7f800000) {
```

```
355                 if (i < 0x00800000)
356                         return ((i == 0) ? fp_zero : fp_subnormal);

358                 return (fp_normal);
359         } else if (i == 0x7f800000) {
360                 return (fp_infinity);
361         } else if (i & 0x400000) {
362                 return (fp_quiet);
363         } else {
364                 return (fp_signaling);
176                         return ((i == 0)? fp_zero : fp_subnormal);
177                 return fp_normal;
365         }
179         else if (i == 0x7f800000)
180                 return fp_infinity;
181         else if (i & 0x400000)
182                 return fp_quiet;
183         else
184                 return fp_signaling;
366 }

368 static enum fp_class_type
369 my_fp_class(double *x)
370 {
371         int i = *(1 + (int *)x) & ~0x80000000;
190         int     i = *(1+(int *)x) & ~0x80000000;

373         if (i < 0x7ff00000) {
374                 if (i < 0x00100000)
375                         return (((i | *(int *)x) == 0) ? fp_zero :
376                             fp_subnormal);

378                 return (fp_normal);
379         } else if (i == 0x7ff00000 && *(int *)x == 0) {
380                 return (fp_infinity);
381         } else if (i & 0x80000) {
382                 return (fp_quiet);
383         } else {
384                 return (fp_signaling);
194                         return (((i | *(int *)x) == 0)? fp_zero : fp_subnormal);
195                 return fp_normal;
385         }
197         else if (i == 0x7ff00000 && *(int *)x == 0)
198                 return fp_infinity;
199         else if (i & 0x80000)
200                 return fp_quiet;
201         else
202                 return fp_signaling;
386 }

388 /*
389  * Inspect a scalar SSE instruction that incurred an invalid operation
390  * exception to determine which type of exception it was.
391  */
392 static enum fex_exception
393 __fex_get_sse_invalid_type(sseinst_t *inst)
394 {
395         enum fp_class_type t1, t2;

397         /* check op2 for signaling nan */
398         t2 = ((int)inst->op & DOUBLE) ? my_fp_class(&inst->op2->d[0]) :
215         t2 = ((int)inst->op & DOUBLE)? my_fp_class(&inst->op2->d[0]) :
399             my_fp_classf(&inst->op2->f[0]);

401 #endif /* ! codereview */
402         if (t2 == fp_signaling)
```



```
403                 return (fex_inv_snan);
217                 return fex_inv_snan;

405         /* eliminate all single-operand instructions */
406         switch (inst->op) {
407         case cvtsd2ss:
408         case cvtss2sd:
409                 /* hmm, this shouldn't have happened */
410                 return ((enum fex_exception)-1);
224                 return (enum fex_exception) -1;

412         case sqrtss:
413         case sqrtsd:
414                 return (fex_inv_sqrt);
228                 return fex_inv_sqrt;

416         case cvtss2si:
417         case cvtsd2si:
418         case cvttss2si:
419         case cvttsd2si:
420         case cvtss2siq:
421         case cvtsd2siq:
422         case cvttss2siq:
423         case cvttsd2siq:
424                 return (fex_inv_int);
238                 return fex_inv_int;
425         default:
426                 break;
427         }

429         /* check op1 for signaling nan */
430         t1 = ((int)inst->op & DOUBLE) ? my_fp_class(&inst->op1->d[0]) :
244         t1 = ((int)inst->op & DOUBLE)? my_fp_class(&inst->op1->d[0]) :
431             my_fp_classf(&inst->op1->f[0]);

433 #endif /* ! codereview */
434         if (t1 == fp_signaling)
435                 return (fex_inv_snan);
246                 return fex_inv_snan;

437         /* check two-operand instructions for other cases */
438         switch (inst->op) {
439         case cmpss:
440         case cmpsd:
441         case minss:
442         case minsd:
443         case maxss:
444         case maxsd:
445         case comiss:
446         case comisd:
447                 return (fex_inv_cmp);
258                 return fex_inv_cmp;

449         case addss:
450         case addsd:
451         case subss:
452         case subsd:

454 #endif /* ! codereview */
455                 if (t1 == fp_infinity && t2 == fp_infinity)
456                         return (fex_inv_isi);

264                         return fex_inv_isi;
458                 break;

460         case mulss:
```

```
        case mulsd:

                if ((t1 == fp_zero && t2 == fp_infinity) || (t2 == fp_zero &&
                    t1 == fp_infinity))
                        return (fex_inv_zmi);

                if ((t1 == fp_zero && t2 == fp_infinity) ||
                    (t2 == fp_zero && t1 == fp_infinity))
                        return fex_inv_zmi;
                break;

        case divss:
        case divsd:

#endif /* ! codereview */
                if (t1 == fp_zero && t2 == fp_zero)
                        return (fex_inv_zdz);

                        return fex_inv_zdz;
                if (t1 == fp_infinity && t2 == fp_infinity)
                        return (fex_inv_idi);

                        return fex_inv_idi;
        default:
                break;
        }

        return ((enum fex_exception)-1);
        return (enum fex_exception)-1;
}

/* inline templates */
extern void sse_cmpeqss(float *, float *, int *);
extern void sse_cmpltss(float *, float *, int *);
extern void sse_cmpless(float *, float *, int *);
extern void sse_cmpunordss(float *, float *, int *);
extern void sse_minss(float *, float *, float *);
extern void sse_maxss(float *, float *, float *);
extern void sse_addss(float *, float *, float *);
extern void sse_subss(float *, float *, float *);
extern void sse_mulss(float *, float *, float *);
extern void sse_divss(float *, float *, float *);
extern void sse_sqrtss(float *, float *);
extern void sse_ucomiss(float *, float *);
extern void sse_comiss(float *, float *);
extern void sse_cvtss2sd(float *, double *);
extern void sse_cvtsi2ss(int *, float *);
extern void sse_cvttss2si(float *, int *);
extern void sse_cvtss2si(float *, int *);

#endif /* ! codereview */
#ifdef __amd64
extern void sse_cvtsi2ssq(long long *, float *);
extern void sse_cvttss2siq(float *, long long *);
extern void sse_cvtss2siq(float *, long long *);
#endif

#endif /* ! codereview */
extern void sse_cmpeqsd(double *, double *, long long *);
extern void sse_cmpltsd(double *, double *, long long *);
extern void sse_cmplesd(double *, double *, long long *);
extern void sse_cmpunordsd(double *, double *, long long *);
extern void sse_minsd(double *, double *, double *);
extern void sse_maxsd(double *, double *, double *);
extern void sse_addsd(double *, double *, double *);
extern void sse_subsd(double *, double *, double *);
```



```
extern void sse_mulsd(double *, double *, double *);
extern void sse_divsd(double *, double *, double *);
extern void sse_sqrtsd(double *, double *);
extern void sse_ucomisd(double *, double *);
extern void sse_comisd(double *, double *);
extern void sse_cvtsd2ss(double *, float *);
extern void sse_cvtsi2sd(int *, double *);
extern void sse_cvttsd2si(double *, int *);
extern void sse_cvtsd2si(double *, int *);

#endif /* ! codereview */
#ifdef __amd64
extern void sse_cvtsi2sdq(long long *, double *);
extern void sse_cvttsd2siq(double *, long long *);
extern void sse_cvtsd2siq(double *, long long *);
#endif

/*
 * Fill in *info with the operands, default untrapped result, and
 * flags produced by a scalar SSE instruction, and return the type
 * of trapped exception (if any).  On entry, the mxcsr must have
 * all exceptions masked and all flags clear.  The same conditions
 * will hold on exit.
 *
 * This routine does not work if the instruction specified by *inst
 * is not a scalar instruction.
 */
enum fex_exception
__fex_get_sse_op(ucontext_t *uap, sseinst_t *inst, fex_info_t *info)
{
        unsigned int e, te, mxcsr, oldmxcsr, subnorm;

        /*
         * Perform the operation with traps disabled and check the
         * exception flags.  If the underflow trap was enabled, also
         * check for an exact subnormal result.
         */
        __fenv_getmxcsr(&oldmxcsr);
        subnorm = 0;

#endif /* ! codereview */
        if ((int)inst->op & DOUBLE) {
                if (inst->op == cvtsi2sd) {
                        info->op1.type = fex_int;
                        info->op1.val.i = inst->op2->i[0];
                        info->op2.type = fex_nodata;
                } else if (inst->op == cvtsi2sdq) {
                        info->op1.type = fex_llong;
                        info->op1.val.l = inst->op2->l[0];
                        info->op2.type = fex_nodata;
                } else if (inst->op == sqrtsd || inst->op == cvtsd2ss ||
                    inst->op == cvttsd2si || inst->op == cvtsd2si || inst->op ==
                    cvttsd2siq || inst->op == cvtsd2siq) {
                    inst->op == cvttsd2si || inst->op == cvtsd2si ||
                    inst->op == cvttsd2siq || inst->op == cvtsd2siq) {
                        info->op1.type = fex_double;
                        info->op1.val.d = inst->op2->d[0];
                        info->op2.type = fex_nodata;
                } else {
                        info->op1.type = fex_double;
                        info->op1.val.d = inst->op1->d[0];
                        info->op2.type = fex_double;
                        info->op2.val.d = inst->op2->d[0];
                }

#endif /* ! codereview */
```

```
                info->res.type = fex_double;

#endif /* ! codereview */
                switch (inst->op) {
                case cmpsd:
                        info->op = fex_cmp;
                        info->res.type = fex_llong;

#endif /* ! codereview */
                        switch (inst->imm & 3) {
                        case 0:
                                sse_cmpeqsd(&info->op1.val.d, &info->op2.val.d,
                                    &info->res.val.l);
                                break;

                        case 1:
                                sse_cmpltsd(&info->op1.val.d, &info->op2.val.d,
                                    &info->res.val.l);
                                break;

                        case 2:
                                sse_cmplesd(&info->op1.val.d, &info->op2.val.d,
                                    &info->res.val.l);
                                break;

                        case 3:
                                sse_cmpunordsd(&info->op1.val.d,
                                    &info->op2.val.d, &info->res.val.l);
                        }

#endif /* ! codereview */
                        if (inst->imm & 4)
                                info->res.val.l ^= 0xffffffffffffffffull;

#endif /* ! codereview */
                        break;

                case minsd:
                        info->op = fex_other;
                        sse_minsd(&info->op1.val.d, &info->op2.val.d,
                            &info->res.val.d);
                        break;

                case maxsd:
                        info->op = fex_other;
                        sse_maxsd(&info->op1.val.d, &info->op2.val.d,
                            &info->res.val.d);
                        break;

                case addsd:
                        info->op = fex_add;
                        sse_addsd(&info->op1.val.d, &info->op2.val.d,
                            &info->res.val.d);

#endif /* ! codereview */
                        if (my_fp_class(&info->res.val.d) == fp_subnormal)
                                subnorm = 1;

#endif /* ! codereview */
                        break;

                case subsd:
                        info->op = fex_sub;
                        sse_subsd(&info->op1.val.d, &info->op2.val.d,
                            &info->res.val.d);
```



```
#endif /* ! codereview */
                        if (my_fp_class(&info->res.val.d) == fp_subnormal)
                                subnorm = 1;

#endif /* ! codereview */
                        break;

                case mulsd:
                        info->op = fex_mul;
                        sse_mulsd(&info->op1.val.d, &info->op2.val.d,
                            &info->res.val.d);

#endif /* ! codereview */
                        if (my_fp_class(&info->res.val.d) == fp_subnormal)
                                subnorm = 1;

#endif /* ! codereview */
                        break;

                case divsd:
                        info->op = fex_div;
                        sse_divsd(&info->op1.val.d, &info->op2.val.d,
                            &info->res.val.d);

#endif /* ! codereview */
                        if (my_fp_class(&info->res.val.d) == fp_subnormal)
                                subnorm = 1;

#endif /* ! codereview */
                        break;

                case sqrtsd:
                        info->op = fex_sqrt;
                        sse_sqrtsd(&info->op1.val.d, &info->res.val.d);
                        break;

                case cvtsd2ss:
                        info->op = fex_cnvt;
                        info->res.type = fex_float;
                        sse_cvtsd2ss(&info->op1.val.d, &info->res.val.f);

#endif /* ! codereview */
                        if (my_fp_classf(&info->res.val.f) == fp_subnormal)
                                subnorm = 1;

#endif /* ! codereview */
                        break;

                case cvtsi2sd:
                        info->op = fex_cnvt;
                        sse_cvtsi2sd(&info->op1.val.i, &info->res.val.d);
                        break;

                case cvttsd2si:
                        info->op = fex_cnvt;
                        info->res.type = fex_int;
                        sse_cvttsd2si(&info->op1.val.d, &info->res.val.i);
                        break;

                case cvtsd2si:
                        info->op = fex_cnvt;
                        info->res.type = fex_int;
                        sse_cvtsd2si(&info->op1.val.d, &info->res.val.i);
                        break;

#ifdef __amd64
```

```
                case cvtsi2sdq:
                        info->op = fex_cnvt;
                        sse_cvtsi2sdq(&info->op1.val.l, &info->res.val.d);
                        break;

                case cvttsd2siq:
                        info->op = fex_cnvt;
                        info->res.type = fex_llong;
                        sse_cvttsd2siq(&info->op1.val.d, &info->res.val.l);
                        break;

                case cvtsd2siq:
                        info->op = fex_cnvt;
                        info->res.type = fex_llong;
                        sse_cvtsd2siq(&info->op1.val.d, &info->res.val.l);
                        break;
#endif

                case ucomisd:
                        info->op = fex_cmp;
                        info->res.type = fex_nodata;
                        sse_ucomisd(&info->op1.val.d, &info->op2.val.d);
                        break;

                case comisd:
                        info->op = fex_cmp;
                        info->res.type = fex_nodata;
                        sse_comisd(&info->op1.val.d, &info->op2.val.d);
                        break;
                default:
                        break;
                }
        } else {
                if (inst->op == cvtsi2ss) {
                        info->op1.type = fex_int;
                        info->op1.val.i = inst->op2->i[0];
                        info->op2.type = fex_nodata;
                } else if (inst->op == cvtsi2ssq) {
                        info->op1.type = fex_llong;
                        info->op1.val.l = inst->op2->l[0];
                        info->op2.type = fex_nodata;
                } else if (inst->op == sqrtss || inst->op == cvtss2sd ||
                    inst->op == cvttss2si || inst->op == cvtss2si || inst->op ==
                    cvttss2siq || inst->op == cvtss2siq) {
                    inst->op == cvttss2si || inst->op == cvtss2si ||
                    inst->op == cvttss2siq || inst->op == cvtss2siq) {
                        info->op1.type = fex_float;
                        info->op1.val.f = inst->op2->f[0];
                        info->op2.type = fex_nodata;
                } else {
                        info->op1.type = fex_float;
                        info->op1.val.f = inst->op1->f[0];
                        info->op2.type = fex_float;
                        info->op2.val.f = inst->op2->f[0];
                }

#endif /* ! codereview */
                info->res.type = fex_float;

#endif /* ! codereview */
                switch (inst->op) {
                case cmpss:
                        info->op = fex_cmp;
                        info->res.type = fex_int;

#endif /* ! codereview */
```



```
                        switch (inst->imm & 3) {
                        case 0:
                                sse_cmpeqss(&info->op1.val.f, &info->op2.val.f,
                                    &info->res.val.i);
                                break;

                        case 1:
                                sse_cmpltss(&info->op1.val.f, &info->op2.val.f,
                                    &info->res.val.i);
                                break;

                        case 2:
                                sse_cmpless(&info->op1.val.f, &info->op2.val.f,
                                    &info->res.val.i);
                                break;

                        case 3:
                                sse_cmpunordss(&info->op1.val.f,
                                    &info->op2.val.f, &info->res.val.i);
                        }

#endif /* ! codereview */
                        if (inst->imm & 4)
                                info->res.val.i ^= 0xffffffffu;

#endif /* ! codereview */
                        break;

                case minss:
                        info->op = fex_other;
                        sse_minss(&info->op1.val.f, &info->op2.val.f,
                            &info->res.val.f);
                        break;

                case maxss:
                        info->op = fex_other;
                        sse_maxss(&info->op1.val.f, &info->op2.val.f,
                            &info->res.val.f);
                        break;

                case addss:
                        info->op = fex_add;
                        sse_addss(&info->op1.val.f, &info->op2.val.f,
                            &info->res.val.f);

#endif /* ! codereview */
                        if (my_fp_classf(&info->res.val.f) == fp_subnormal)
                                subnorm = 1;

#endif /* ! codereview */
                        break;

                case subss:
                        info->op = fex_sub;
                        sse_subss(&info->op1.val.f, &info->op2.val.f,
                            &info->res.val.f);

#endif /* ! codereview */
                        if (my_fp_classf(&info->res.val.f) == fp_subnormal)
                                subnorm = 1;

#endif /* ! codereview */
                        break;

                case mulss:
                        info->op = fex_mul;
```

```
847                         sse_mulss(&info->op1.val.f, &info->op2.val.f,
848                             &info->res.val.f);

850 #endif /* ! codereview */
851                         if (my_fp_classf(&info->res.val.f) == fp_subnormal)
852                                 subnorm = 1;

854 #endif /* ! codereview */
855                         break;

857                 case divss:
858                         info->op = fex_div;
859                         sse_divss(&info->op1.val.f, &info->op2.val.f,
860                             &info->res.val.f);

862 #endif /* ! codereview */
863                         if (my_fp_classf(&info->res.val.f) == fp_subnormal)
864                                 subnorm = 1;

866 #endif /* ! codereview */
867                         break;

869                 case sqrtss:
870                         info->op = fex_sqrt;
871                         sse_sqrtss(&info->op1.val.f, &info->res.val.f);
872                         break;

874                 case cvtss2sd:
875                         info->op = fex_cnvt;
876                         info->res.type = fex_double;
877                         sse_cvtss2sd(&info->op1.val.f, &info->res.val.d);
878                         break;

880                 case cvtsi2ss:
881                         info->op = fex_cnvt;
882                         sse_cvtsi2ss(&info->op1.val.i, &info->res.val.f);
883                         break;

885                 case cvttss2si:
886                         info->op = fex_cnvt;
887                         info->res.type = fex_int;
888                         sse_cvttss2si(&info->op1.val.f, &info->res.val.i);
889                         break;

891                 case cvtss2si:
892                         info->op = fex_cnvt;
893                         info->res.type = fex_int;
894                         sse_cvtss2si(&info->op1.val.f, &info->res.val.i);
895                         break;

897 #ifdef __amd64
898                 case cvtsi2ssq:
899                         info->op = fex_cnvt;
900                         sse_cvtsi2ssq(&info->op1.val.l, &info->res.val.f);
901                         break;

903                 case cvttss2siq:
904                         info->op = fex_cnvt;
905                         info->res.type = fex_llong;
906                         sse_cvttss2siq(&info->op1.val.f, &info->res.val.l);
907                         break;

909                 case cvtss2siq:
910                         info->op = fex_cnvt;
911                         info->res.type = fex_llong;
912                         sse_cvtss2siq(&info->op1.val.f, &info->res.val.l);
```



```
913                         break;
914 #endif

916                 case ucomiss:
917                         info->op = fex_cmp;
918                         info->res.type = fex_nodata;
919                         sse_ucomiss(&info->op1.val.f, &info->op2.val.f);
920                         break;

922                 case comiss:
923                         info->op = fex_cmp;
924                         info->res.type = fex_nodata;
925                         sse_comiss(&info->op1.val.f, &info->op2.val.f);
926                         break;
927                 default:
928                         break;
929                 }
930         }

932 #endif /* ! codereview */
933         __fenv_getmxcsr(&mxcsr);
934         info->flags = mxcsr & 0x3d;
935         __fenv_setmxcsr(&oldmxcsr);

937         /* determine which exception would have been trapped */
938         te = ~(uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state.mxcsr >> 7) &
939             0x3d;
326         te = ~(uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state.mxcsr
327             >> 7) & 0x3d;
940         e = mxcsr & te;

942 #endif /* ! codereview */
943         if (e & FE_INVALID)
944                 return (__fex_get_sse_invalid_type(inst));

329                 return __fex_get_sse_invalid_type(inst);
946         if (e & FE_DIVBYZERO)
947                 return (fex_division);

331                 return fex_division;
949         if (e & FE_OVERFLOW)
950                 return (fex_overflow);

333                 return fex_overflow;
952         if ((e & FE_UNDERFLOW) || (subnorm && (te & FE_UNDERFLOW)))
953                 return (fex_underflow);

335                 return fex_underflow;
955         if (e & FE_INEXACT)
956                 return (fex_inexact);

958         return ((enum fex_exception)-1);
337                 return fex_inexact;
338         return (enum fex_exception)-1;
959 }

961 /*
962  * Emulate a SIMD SSE instruction to determine which exceptions occur
963  * in each part.  For i = 0, 1, 2, and 3, set e[i] to indicate the
964  * trapped exception that would occur if the i-th part of the SIMD
965  * instruction were executed in isolation; set e[i] to -1 if no
966  * trapped exception would occur in this part.  Also fill in info[i]
967  * with the corresponding operands, default untrapped result, and
968  * flags.
969  *
970  * This routine does not work if the instruction specified by *inst
```

```
 * is not a SIMD instruction.
 */
void
__fex_get_simd_op(ucontext_t *uap, sseinst_t *inst, enum fex_exception *e,
    fex_info_t *info)
{
        sseinst_t dummy;
        int i;

        e[0] = e[1] = e[2] = e[3] = -1;

        /* perform each part of the SIMD operation */
        switch (inst->op) {
        case cmpps:
                dummy.op = cmpss;
                dummy.imm = inst->imm;
#endif /* ! codereview */
                for (i = 0; i < 4; i++) {
                        dummy.op1 = (sseoperand_t *)&inst->op1->f[i];
                        dummy.op2 = (sseoperand_t *)&inst->op2->f[i];
                        e[i] = __fex_get_sse_op(uap, &dummy, &info[i]);
                }
#endif /* ! codereview */
                break;

        case minps:
                dummy.op = minss;
#endif /* ! codereview */
                for (i = 0; i < 4; i++) {
                        dummy.op1 = (sseoperand_t *)&inst->op1->f[i];
                        dummy.op2 = (sseoperand_t *)&inst->op2->f[i];
                        e[i] = __fex_get_sse_op(uap, &dummy, &info[i]);
                }
#endif /* ! codereview */
                break;

        case maxps:
                dummy.op = maxss;
#endif /* ! codereview */
                for (i = 0; i < 4; i++) {
                        dummy.op1 = (sseoperand_t *)&inst->op1->f[i];
                        dummy.op2 = (sseoperand_t *)&inst->op2->f[i];
                        e[i] = __fex_get_sse_op(uap, &dummy, &info[i]);
                }
#endif /* ! codereview */
                break;

        case addps:
                dummy.op = addss;
#endif /* ! codereview */
                for (i = 0; i < 4; i++) {
                        dummy.op1 = (sseoperand_t *)&inst->op1->f[i];
                        dummy.op2 = (sseoperand_t *)&inst->op2->f[i];
                        e[i] = __fex_get_sse_op(uap, &dummy, &info[i]);
                }
#endif /* ! codereview */
                break;
```



```
        case subps:
                dummy.op = subss;
#endif /* ! codereview */
                for (i = 0; i < 4; i++) {
                        dummy.op1 = (sseoperand_t *)&inst->op1->f[i];
                        dummy.op2 = (sseoperand_t *)&inst->op2->f[i];
                        e[i] = __fex_get_sse_op(uap, &dummy, &info[i]);
                }
#endif /* ! codereview */
                break;

        case mulps:
                dummy.op = mulss;
#endif /* ! codereview */
                for (i = 0; i < 4; i++) {
                        dummy.op1 = (sseoperand_t *)&inst->op1->f[i];
                        dummy.op2 = (sseoperand_t *)&inst->op2->f[i];
                        e[i] = __fex_get_sse_op(uap, &dummy, &info[i]);
                }
#endif /* ! codereview */
                break;

        case divps:
                dummy.op = divss;
#endif /* ! codereview */
                for (i = 0; i < 4; i++) {
                        dummy.op1 = (sseoperand_t *)&inst->op1->f[i];
                        dummy.op2 = (sseoperand_t *)&inst->op2->f[i];
                        e[i] = __fex_get_sse_op(uap, &dummy, &info[i]);
                }
#endif /* ! codereview */
                break;

        case sqrtps:
                dummy.op = sqrtss;
#endif /* ! codereview */
                for (i = 0; i < 4; i++) {
                        dummy.op1 = (sseoperand_t *)&inst->op1->f[i];
                        dummy.op2 = (sseoperand_t *)&inst->op2->f[i];
                        e[i] = __fex_get_sse_op(uap, &dummy, &info[i]);
                }
#endif /* ! codereview */
                break;

        case cvtdq2ps:
                dummy.op = cvtsi2ss;
#endif /* ! codereview */
                for (i = 0; i < 4; i++) {
                        dummy.op1 = (sseoperand_t *)&inst->op1->f[i];
                        dummy.op2 = (sseoperand_t *)&inst->op2->i[i];
                        e[i] = __fex_get_sse_op(uap, &dummy, &info[i]);
                }
#endif /* ! codereview */
                break;

        case cvttps2dq:
```

```
                dummy.op = cvttss2si;

#endif /* ! codereview */
                for (i = 0; i < 4; i++) {
                        dummy.op1 = (sseoperand_t *)&inst->op1->i[i];
                        dummy.op2 = (sseoperand_t *)&inst->op2->f[i];
                        e[i] = __fex_get_sse_op(uap, &dummy, &info[i]);
                }

#endif /* ! codereview */
                break;

        case cvtps2dq:
                dummy.op = cvtss2si;

#endif /* ! codereview */
                for (i = 0; i < 4; i++) {
                        dummy.op1 = (sseoperand_t *)&inst->op1->i[i];
                        dummy.op2 = (sseoperand_t *)&inst->op2->f[i];
                        e[i] = __fex_get_sse_op(uap, &dummy, &info[i]);
                }

#endif /* ! codereview */
                break;

        case cvtpi2ps:
                dummy.op = cvtsi2ss;

#endif /* ! codereview */
                for (i = 0; i < 2; i++) {
                        dummy.op1 = (sseoperand_t *)&inst->op1->f[i];
                        dummy.op2 = (sseoperand_t *)&inst->op2->i[i];
                        e[i] = __fex_get_sse_op(uap, &dummy, &info[i]);
                }

#endif /* ! codereview */
                break;

        case cvttps2pi:
                dummy.op = cvttss2si;

#endif /* ! codereview */
                for (i = 0; i < 2; i++) {
                        dummy.op1 = (sseoperand_t *)&inst->op1->i[i];
                        dummy.op2 = (sseoperand_t *)&inst->op2->f[i];
                        e[i] = __fex_get_sse_op(uap, &dummy, &info[i]);
                }

#endif /* ! codereview */
                break;

        case cvtps2pi:
                dummy.op = cvtss2si;

#endif /* ! codereview */
                for (i = 0; i < 2; i++) {
                        dummy.op1 = (sseoperand_t *)&inst->op1->i[i];
                        dummy.op2 = (sseoperand_t *)&inst->op2->f[i];
                        e[i] = __fex_get_sse_op(uap, &dummy, &info[i]);
                }

#endif /* ! codereview */
                break;

        case cmppd:
                dummy.op = cmpsd;
```



```
                dummy.imm = inst->imm;

#endif /* ! codereview */
                for (i = 0; i < 2; i++) {
                        dummy.op1 = (sseoperand_t *)&inst->op1->d[i];
                        dummy.op2 = (sseoperand_t *)&inst->op2->d[i];
                        e[i] = __fex_get_sse_op(uap, &dummy, &info[i]);
                }

#endif /* ! codereview */
                break;

        case minpd:
                dummy.op = minsd;

#endif /* ! codereview */
                for (i = 0; i < 2; i++) {
                        dummy.op1 = (sseoperand_t *)&inst->op1->d[i];
                        dummy.op2 = (sseoperand_t *)&inst->op2->d[i];
                        e[i] = __fex_get_sse_op(uap, &dummy, &info[i]);
                }

#endif /* ! codereview */
                break;

        case maxpd:
                dummy.op = maxsd;

#endif /* ! codereview */
                for (i = 0; i < 2; i++) {
                        dummy.op1 = (sseoperand_t *)&inst->op1->d[i];
                        dummy.op2 = (sseoperand_t *)&inst->op2->d[i];
                        e[i] = __fex_get_sse_op(uap, &dummy, &info[i]);
                }

#endif /* ! codereview */
                break;

        case addpd:
                dummy.op = addsd;

#endif /* ! codereview */
                for (i = 0; i < 2; i++) {
                        dummy.op1 = (sseoperand_t *)&inst->op1->d[i];
                        dummy.op2 = (sseoperand_t *)&inst->op2->d[i];
                        e[i] = __fex_get_sse_op(uap, &dummy, &info[i]);
                }

#endif /* ! codereview */
                break;

        case subpd:
                dummy.op = subsd;

#endif /* ! codereview */
                for (i = 0; i < 2; i++) {
                        dummy.op1 = (sseoperand_t *)&inst->op1->d[i];
                        dummy.op2 = (sseoperand_t *)&inst->op2->d[i];
                        e[i] = __fex_get_sse_op(uap, &dummy, &info[i]);
                }

#endif /* ! codereview */
                break;

        case mulpd:
                dummy.op = mulsd;
```

```
#endif /* ! codereview */
                for (i = 0; i < 2; i++) {
                        dummy.op1 = (sseoperand_t *)&inst->op1->d[i];
                        dummy.op2 = (sseoperand_t *)&inst->op2->d[i];
                        e[i] = __fex_get_sse_op(uap, &dummy, &info[i]);
                }

#endif /* ! codereview */
                break;

        case divpd:
                dummy.op = divsd;

#endif /* ! codereview */
                for (i = 0; i < 2; i++) {
                        dummy.op1 = (sseoperand_t *)&inst->op1->d[i];
                        dummy.op2 = (sseoperand_t *)&inst->op2->d[i];
                        e[i] = __fex_get_sse_op(uap, &dummy, &info[i]);
                }

#endif /* ! codereview */
                break;

        case sqrtpd:
                dummy.op = sqrtsd;

#endif /* ! codereview */
                for (i = 0; i < 2; i++) {
                        dummy.op1 = (sseoperand_t *)&inst->op1->d[i];
                        dummy.op2 = (sseoperand_t *)&inst->op2->d[i];
                        e[i] = __fex_get_sse_op(uap, &dummy, &info[i]);
                }

#endif /* ! codereview */
                break;

        case cvtpi2pd:
        case cvtdq2pd:
                dummy.op = cvtsi2sd;

#endif /* ! codereview */
                for (i = 0; i < 2; i++) {
                        dummy.op1 = (sseoperand_t *)&inst->op1->d[i];
                        dummy.op2 = (sseoperand_t *)&inst->op2->i[i];
                        e[i] = __fex_get_sse_op(uap, &dummy, &info[i]);
                }

#endif /* ! codereview */
                break;

        case cvttpd2pi:
        case cvttpd2dq:
                dummy.op = cvttsd2si;

#endif /* ! codereview */
                for (i = 0; i < 2; i++) {
                        dummy.op1 = (sseoperand_t *)&inst->op1->i[i];
                        dummy.op2 = (sseoperand_t *)&inst->op2->d[i];
                        e[i] = __fex_get_sse_op(uap, &dummy, &info[i]);
                }

#endif /* ! codereview */
                break;

        case cvtpd2pi:
```



```
        case cvtpd2dq:
                dummy.op = cvtsd2si;

#endif /* ! codereview */
                for (i = 0; i < 2; i++) {
                        dummy.op1 = (sseoperand_t *)&inst->op1->i[i];
                        dummy.op2 = (sseoperand_t *)&inst->op2->d[i];
                        e[i] = __fex_get_sse_op(uap, &dummy, &info[i]);
                }

#endif /* ! codereview */
                break;

        case cvtps2pd:
                dummy.op = cvtss2sd;

#endif /* ! codereview */
                for (i = 0; i < 2; i++) {
                        dummy.op1 = (sseoperand_t *)&inst->op1->d[i];
                        dummy.op2 = (sseoperand_t *)&inst->op2->f[i];
                        e[i] = __fex_get_sse_op(uap, &dummy, &info[i]);
                }

#endif /* ! codereview */
                break;

        case cvtpd2ps:
                dummy.op = cvtsd2ss;

#endif /* ! codereview */
                for (i = 0; i < 2; i++) {
                        dummy.op1 = (sseoperand_t *)&inst->op1->f[i];
                        dummy.op2 = (sseoperand_t *)&inst->op2->d[i];
                        e[i] = __fex_get_sse_op(uap, &dummy, &info[i]);
                }

#endif /* ! codereview */
        default:
                break;
        }
}

/*
 * Store the result value from *info in the destination of the scalar
 * SSE instruction specified by *inst.  If no result is given but the
 * exception is underflow or overflow, supply the default trapped result.
 *
 * This routine does not work if the instruction specified by *inst
 * is not a scalar instruction.
 */
void
__fex_st_sse_result(ucontext_t *uap, sseinst_t *inst, enum fex_exception e,
    fex_info_t *info)
{
        int i = 0;
        long long l = 0L;
        long long       l = 0L;;
        float f = 0.0, fscl;
        double d = 0.0L, dscl;

        /*
         * for compares that write eflags, just set the flags
         * to indicate "unordered"
         */
        if (inst->op == ucomiss || inst->op == comiss || inst->op == ucomisd ||
            inst->op == comisd) {
```

```
        /* for compares that write eflags, just set the flags
           to indicate "unordered" */
        if (inst->op == ucomiss || inst->op == comiss ||
            inst->op == ucomisd || inst->op == comisd) {
                uap->uc_mcontext.gregs[REG_PS] |= 0x45;
                return;
        }

        /*
         * if info doesn't specify a result value, try to generate
         * the default trapped result
         */
        /* if info doesn't specify a result value, try to generate
           the default trapped result */
        if (info->res.type == fex_nodata) {
                /* set scale factors for exponent wrapping */
                switch (e) {
                case fex_overflow:
                        fscl = 1.262177448e-29f;                /* 2^-96 */
                        dscl = 6.441148769597133308e-232;       /* 2^-768 */
                        break;

                case fex_underflow:
                        fscl = 7.922816251e+28f;                /* 2^96 */
                        dscl = 1.552518092300708935e+231;       /* 2^768 */
                        break;

                default:
                        (void) __fex_get_sse_op(uap, inst, info);

#endif /* ! codereview */
                        if (info->res.type == fex_nodata)
                                return;

#endif /* ! codereview */
                        goto stuff;
                }

                /* generate the wrapped result */
                if (inst->op == cvtsd2ss) {
                        info->op1.type = fex_double;
                        info->op1.val.d = inst->op2->d[0];
                        info->op2.type = fex_nodata;
                        info->res.type = fex_float;
                        info->res.val.f = (float)(fscl * (fscl *
                            info->op1.val.d));
                } else if ((int)inst->op & DOUBLE) {
                        info->op1.type = fex_double;
                        info->op1.val.d = inst->op1->d[0];
                        info->op2.type = fex_double;
                        info->op2.val.d = inst->op2->d[0];
                        info->res.type = fex_double;

#endif /* ! codereview */
                        switch (inst->op) {
                        case addsd:
                                info->res.val.d = dscl * (dscl *
                                    info->op1.val.d + dscl * info->op2.val.d);
                                break;

                        case subsd:
                                info->res.val.d = dscl * (dscl *
                                    info->op1.val.d - dscl * info->op2.val.d);
                                break;

                        case mulsd:
```



```
                                info->res.val.d = (dscl * info->op1.val.d) *
                                    (dscl * info->op2.val.d);
                                break;

                        case divsd:
                                info->res.val.d = (dscl * info->op1.val.d) /
                                    (info->op2.val.d / dscl);
                                break;

                        default:
                                return;
                        }
                } else {
                        info->op1.type = fex_float;
                        info->op1.val.f = inst->op1->f[0];
                        info->op2.type = fex_float;
                        info->op2.val.f = inst->op2->f[0];
                        info->res.type = fex_float;

#endif /* ! codereview */
                        switch (inst->op) {
                        case addss:
                                info->res.val.f = fscl * (fscl *
                                    info->op1.val.f + fscl * info->op2.val.f);
                                break;

                        case subss:
                                info->res.val.f = fscl * (fscl *
                                    info->op1.val.f - fscl * info->op2.val.f);
                                break;

                        case mulss:
                                info->res.val.f = (fscl * info->op1.val.f) *
                                    (fscl * info->op2.val.f);
                                break;

                        case divss:
                                info->res.val.f = (fscl * info->op1.val.f) /
                                    (info->op2.val.f / fscl);
                                break;

                        default:
                                return;
                        }
                }
        }

        /* put the result in the destination */
stuff:
        if (inst->op == cmpss || inst->op == cvttss2si || inst->op ==
            cvtss2si || inst->op == cvttsd2si || inst->op == cvtsd2si) {
        if (inst->op == cmpss || inst->op == cvttss2si || inst->op == cvtss2si
            || inst->op == cvttsd2si || inst->op == cvtsd2si) {
                switch (info->res.type) {
                case fex_int:
                        i = info->res.val.i;
                        break;

                case fex_llong:
                        i = info->res.val.l;
                        break;

                case fex_float:
                        i = info->res.val.f;
                        break;
```

```
                case fex_double:
                        i = info->res.val.d;
                        break;

                case fex_ldouble:
                        i = info->res.val.q;
                        break;

                default:
                        break;
                }

#endif /* ! codereview */
                inst->op1->i[0] = i;
        } else if (inst->op == cmpsd || inst->op == cvttss2siq || inst->op ==
            cvtss2siq || inst->op == cvttsd2siq || inst->op == cvtsd2siq) {
        } else if (inst->op == cmpsd || inst->op == cvttss2siq ||
            inst->op == cvtss2siq || inst->op == cvttsd2siq ||
            inst->op == cvtsd2siq) {
                switch (info->res.type) {
                case fex_int:
                        l = info->res.val.i;
                        break;

                case fex_llong:
                        l = info->res.val.l;
                        break;

                case fex_float:
                        l = info->res.val.f;
                        break;

                case fex_double:
                        l = info->res.val.d;
                        break;

                case fex_ldouble:
                        l = info->res.val.q;
                        break;

                default:
                        break;
                }

#endif /* ! codereview */
                inst->op1->l[0] = l;
        } else if ((((int)inst->op & DOUBLE) && inst->op != cvtsd2ss) ||
            inst->op == cvtss2sd) {
                switch (info->res.type) {
                case fex_int:
                        d = info->res.val.i;
                        break;

                case fex_llong:
                        d = info->res.val.l;
                        break;

                case fex_float:
                        d = info->res.val.f;
                        break;

                case fex_double:
                        d = info->res.val.d;
                        break;

                case fex_ldouble:
```



```
                        d = info->res.val.q;
                        break;

                default:
                        break;
                }

#endif /* ! codereview */
                inst->op1->d[0] = d;
        } else {
                switch (info->res.type) {
                case fex_int:
                        f = info->res.val.i;
                        break;

                case fex_llong:
                        f = info->res.val.l;
                        break;

                case fex_float:
                        f = info->res.val.f;
                        break;

                case fex_double:
                        f = info->res.val.d;
                        break;

                case fex_ldouble:
                        f = info->res.val.q;
                        break;

                default:
                        break;
                }

#endif /* ! codereview */
                inst->op1->f[0] = f;
        }
}

/*
 * Store the results from a SIMD instruction.  For each i, store
 * the result value from info[i] in the i-th part of the destination
 * of the SIMD SSE instruction specified by *inst.  If no result
 * is given but the exception indicated by e[i] is underflow or
 * overflow, supply the default trapped result.
 *
 * This routine does not work if the instruction specified by *inst
 * is not a SIMD instruction.
 */
void
__fex_st_simd_result(ucontext_t *uap, sseinst_t *inst, enum fex_exception *e,
    fex_info_t *info)
{
        sseinst_t dummy;
        int i;

        /* store each part */
        switch (inst->op) {
        case cmpps:
                dummy.op = cmpss;
                dummy.imm = inst->imm;

#endif /* ! codereview */
                for (i = 0; i < 4; i++) {
                        dummy.op1 = (sseoperand_t *)&inst->op1->f[i];
```

```
                        dummy.op2 = (sseoperand_t *)&inst->op2->f[i];
                        __fex_st_sse_result(uap, &dummy, e[i], &info[i]);
                }

#endif /* ! codereview */
                break;

        case minps:
                dummy.op = minss;

#endif /* ! codereview */
                for (i = 0; i < 4; i++) {
                        dummy.op1 = (sseoperand_t *)&inst->op1->f[i];
                        dummy.op2 = (sseoperand_t *)&inst->op2->f[i];
                        __fex_st_sse_result(uap, &dummy, e[i], &info[i]);
                }

#endif /* ! codereview */
                break;

        case maxps:
                dummy.op = maxss;

#endif /* ! codereview */
                for (i = 0; i < 4; i++) {
                        dummy.op1 = (sseoperand_t *)&inst->op1->f[i];
                        dummy.op2 = (sseoperand_t *)&inst->op2->f[i];
                        __fex_st_sse_result(uap, &dummy, e[i], &info[i]);
                }

#endif /* ! codereview */
                break;

        case addps:
                dummy.op = addss;

#endif /* ! codereview */
                for (i = 0; i < 4; i++) {
                        dummy.op1 = (sseoperand_t *)&inst->op1->f[i];
                        dummy.op2 = (sseoperand_t *)&inst->op2->f[i];
                        __fex_st_sse_result(uap, &dummy, e[i], &info[i]);
                }

#endif /* ! codereview */
                break;

        case subps:
                dummy.op = subss;

#endif /* ! codereview */
                for (i = 0; i < 4; i++) {
                        dummy.op1 = (sseoperand_t *)&inst->op1->f[i];
                        dummy.op2 = (sseoperand_t *)&inst->op2->f[i];
                        __fex_st_sse_result(uap, &dummy, e[i], &info[i]);
                }

#endif /* ! codereview */
                break;

        case mulps:
                dummy.op = mulss;

#endif /* ! codereview */
                for (i = 0; i < 4; i++) {
                        dummy.op1 = (sseoperand_t *)&inst->op1->f[i];
                        dummy.op2 = (sseoperand_t *)&inst->op2->f[i];
```



```
                        __fex_st_sse_result(uap, &dummy, e[i], &info[i]);
                }

#endif /* ! codereview */
                break;

        case divps:
                dummy.op = divss;

#endif /* ! codereview */
                for (i = 0; i < 4; i++) {
                        dummy.op1 = (sseoperand_t *)&inst->op1->f[i];
                        dummy.op2 = (sseoperand_t *)&inst->op2->f[i];
                        __fex_st_sse_result(uap, &dummy, e[i], &info[i]);
                }

#endif /* ! codereview */
                break;

        case sqrtps:
                dummy.op = sqrtss;

#endif /* ! codereview */
                for (i = 0; i < 4; i++) {
                        dummy.op1 = (sseoperand_t *)&inst->op1->f[i];
                        dummy.op2 = (sseoperand_t *)&inst->op2->f[i];
                        __fex_st_sse_result(uap, &dummy, e[i], &info[i]);
                }

#endif /* ! codereview */
                break;

        case cvtdq2ps:
                dummy.op = cvtsi2ss;

#endif /* ! codereview */
                for (i = 0; i < 4; i++) {
                        dummy.op1 = (sseoperand_t *)&inst->op1->f[i];
                        dummy.op2 = (sseoperand_t *)&inst->op2->i[i];
                        __fex_st_sse_result(uap, &dummy, e[i], &info[i]);
                }

#endif /* ! codereview */
                break;

        case cvttps2dq:
                dummy.op = cvttss2si;

#endif /* ! codereview */
                for (i = 0; i < 4; i++) {
                        dummy.op1 = (sseoperand_t *)&inst->op1->i[i];
                        dummy.op2 = (sseoperand_t *)&inst->op2->f[i];
                        __fex_st_sse_result(uap, &dummy, e[i], &info[i]);
                }

#endif /* ! codereview */
                break;

        case cvtps2dq:
                dummy.op = cvtss2si;

#endif /* ! codereview */
                for (i = 0; i < 4; i++) {
                        dummy.op1 = (sseoperand_t *)&inst->op1->i[i];
                        dummy.op2 = (sseoperand_t *)&inst->op2->f[i];
                        __fex_st_sse_result(uap, &dummy, e[i], &info[i]);
```

```
                }

#endif /* ! codereview */
                break;

        case cvtpi2ps:
                dummy.op = cvtsi2ss;

#endif /* ! codereview */
                for (i = 0; i < 2; i++) {
                        dummy.op1 = (sseoperand_t *)&inst->op1->f[i];
                        dummy.op2 = (sseoperand_t *)&inst->op2->i[i];
                        __fex_st_sse_result(uap, &dummy, e[i], &info[i]);
                }

#endif /* ! codereview */
                break;

        case cvttps2pi:
                dummy.op = cvttss2si;

#endif /* ! codereview */
                for (i = 0; i < 2; i++) {
                        dummy.op1 = (sseoperand_t *)&inst->op1->i[i];
                        dummy.op2 = (sseoperand_t *)&inst->op2->f[i];
                        __fex_st_sse_result(uap, &dummy, e[i], &info[i]);
                }

#endif /* ! codereview */
                break;

        case cvtps2pi:
                dummy.op = cvtss2si;

#endif /* ! codereview */
                for (i = 0; i < 2; i++) {
                        dummy.op1 = (sseoperand_t *)&inst->op1->i[i];
                        dummy.op2 = (sseoperand_t *)&inst->op2->f[i];
                        __fex_st_sse_result(uap, &dummy, e[i], &info[i]);
                }

#endif /* ! codereview */
                break;

        case cmppd:
                dummy.op = cmpsd;
                dummy.imm = inst->imm;

#endif /* ! codereview */
                for (i = 0; i < 2; i++) {
                        dummy.op1 = (sseoperand_t *)&inst->op1->d[i];
                        dummy.op2 = (sseoperand_t *)&inst->op2->d[i];
                        __fex_st_sse_result(uap, &dummy, e[i], &info[i]);
                }

#endif /* ! codereview */
                break;

        case minpd:
                dummy.op = minsd;

#endif /* ! codereview */
                for (i = 0; i < 2; i++) {
                        dummy.op1 = (sseoperand_t *)&inst->op1->d[i];
                        dummy.op2 = (sseoperand_t *)&inst->op2->d[i];
                        __fex_st_sse_result(uap, &dummy, e[i], &info[i]);
```



```
                }

#endif /* ! codereview */
                break;

        case maxpd:
                dummy.op = maxsd;

#endif /* ! codereview */
                for (i = 0; i < 2; i++) {
                        dummy.op1 = (sseoperand_t *)&inst->op1->d[i];
                        dummy.op2 = (sseoperand_t *)&inst->op2->d[i];
                        __fex_st_sse_result(uap, &dummy, e[i], &info[i]);
                }

#endif /* ! codereview */
                break;

        case addpd:
                dummy.op = addsd;

#endif /* ! codereview */
                for (i = 0; i < 2; i++) {
                        dummy.op1 = (sseoperand_t *)&inst->op1->d[i];
                        dummy.op2 = (sseoperand_t *)&inst->op2->d[i];
                        __fex_st_sse_result(uap, &dummy, e[i], &info[i]);
                }

#endif /* ! codereview */
                break;

        case subpd:
                dummy.op = subsd;

#endif /* ! codereview */
                for (i = 0; i < 2; i++) {
                        dummy.op1 = (sseoperand_t *)&inst->op1->d[i];
                        dummy.op2 = (sseoperand_t *)&inst->op2->d[i];
                        __fex_st_sse_result(uap, &dummy, e[i], &info[i]);
                }

#endif /* ! codereview */
                break;

        case mulpd:
                dummy.op = mulsd;

#endif /* ! codereview */
                for (i = 0; i < 2; i++) {
                        dummy.op1 = (sseoperand_t *)&inst->op1->d[i];
                        dummy.op2 = (sseoperand_t *)&inst->op2->d[i];
                        __fex_st_sse_result(uap, &dummy, e[i], &info[i]);
                }

#endif /* ! codereview */
                break;

        case divpd:
                dummy.op = divsd;

#endif /* ! codereview */
                for (i = 0; i < 2; i++) {
                        dummy.op1 = (sseoperand_t *)&inst->op1->d[i];
                        dummy.op2 = (sseoperand_t *)&inst->op2->d[i];
                        __fex_st_sse_result(uap, &dummy, e[i], &info[i]);
                }
```

```
#endif /* ! codereview */
                break;

        case sqrtpd:
                dummy.op = sqrtsd;

#endif /* ! codereview */
                for (i = 0; i < 2; i++) {
                        dummy.op1 = (sseoperand_t *)&inst->op1->d[i];
                        dummy.op2 = (sseoperand_t *)&inst->op2->d[i];
                        __fex_st_sse_result(uap, &dummy, e[i], &info[i]);
                }

#endif /* ! codereview */
                break;

        case cvtpi2pd:
        case cvtdq2pd:
                dummy.op = cvtsi2sd;

#endif /* ! codereview */
                for (i = 0; i < 2; i++) {
                        dummy.op1 = (sseoperand_t *)&inst->op1->d[i];
                        dummy.op2 = (sseoperand_t *)&inst->op2->i[i];
                        __fex_st_sse_result(uap, &dummy, e[i], &info[i]);
                }

#endif /* ! codereview */
                break;

        case cvttpd2pi:
        case cvttpd2dq:
                dummy.op = cvttsd2si;

#endif /* ! codereview */
                for (i = 0; i < 2; i++) {
                        dummy.op1 = (sseoperand_t *)&inst->op1->i[i];
                        dummy.op2 = (sseoperand_t *)&inst->op2->d[i];
                        __fex_st_sse_result(uap, &dummy, e[i], &info[i]);
                }

#endif /* ! codereview */
                /* for cvttpd2dq, zero the high 64 bits of the destination */
                if (inst->op == cvttpd2dq)
                        inst->op1->l[1] = 0ll;

#endif /* ! codereview */
                break;

        case cvtpd2pi:
        case cvtpd2dq:
                dummy.op = cvtsd2si;

#endif /* ! codereview */
                for (i = 0; i < 2; i++) {
                        dummy.op1 = (sseoperand_t *)&inst->op1->i[i];
                        dummy.op2 = (sseoperand_t *)&inst->op2->d[i];
                        __fex_st_sse_result(uap, &dummy, e[i], &info[i]);
                }

#endif /* ! codereview */
                /* for cvtpd2dq, zero the high 64 bits of the destination */
                if (inst->op == cvtpd2dq)
                        inst->op1->l[1] = 0ll;
```



```
#endif /* ! codereview */
                break;

        case cvtps2pd:
                dummy.op = cvtss2sd;

#endif /* ! codereview */
                for (i = 0; i < 2; i++) {
                        dummy.op1 = (sseoperand_t *)&inst->op1->d[i];
                        dummy.op2 = (sseoperand_t *)&inst->op2->f[i];
                        __fex_st_sse_result(uap, &dummy, e[i], &info[i]);
                }

#endif /* ! codereview */
                break;

        case cvtpd2ps:
                dummy.op = cvtsd2ss;

#endif /* ! codereview */
                for (i = 0; i < 2; i++) {
                        dummy.op1 = (sseoperand_t *)&inst->op1->f[i];
                        dummy.op2 = (sseoperand_t *)&inst->op2->d[i];
                        __fex_st_sse_result(uap, &dummy, e[i], &info[i]);
                }

#endif /* ! codereview */
                /* zero the high 64 bits of the destination */
                inst->op1->l[1] = 0ll;

        default:
                break;
        }
}
```

```
**********************************************************
    7776 Thu Jun  6 20:40:06 2019
new/usr/src/lib/libm/common/m9x/__fex_sym.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#include <elf.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <procfs.h>
#include <string.h>
#include <sys/stat.h>

#if defined(__sparcv9) || defined(__amd64)

#define Elf_Ehdr                Elf64_Ehdr
#define Elf_Phdr                Elf64_Phdr
#define Elf_Shdr                Elf64_Shdr
#define Elf_Sym                 Elf64_Sym
#define ELF_ST_BIND             ELF64_ST_BIND
#define ELF_ST_TYPE             ELF64_ST_TYPE

#else

#define Elf_Ehdr                Elf32_Ehdr
#define Elf_Phdr                Elf32_Phdr
#define Elf_Shdr                Elf32_Shdr
#define Elf_Sym                 Elf32_Sym
#define ELF_ST_BIND             ELF32_ST_BIND
#define ELF_ST_TYPE             ELF32_ST_TYPE

#endif /* __sparcv9 */

/* semi-permanent data established by __fex_sym_init */
```



```
static prmap_t *pm = NULL;              /* prmap_t array */
static int npm = 0;                     /* number of entries in pm */

/* transient data modified by __fex_sym */
static prmap_t *lpm = NULL;             /* prmap_t found in last call */
static Elf_Phdr *ph = NULL;             /* program header array */
static int phsize = 0;                  /* size of ph */
static int nph;                         /* number of entries in ph */
static char *stbuf = NULL;              /* symbol and string table buffer */
static int stbufsize = 0;               /* size of stbuf */
static int stoffset;                    /* offset of string table in stbuf */
static int nsyms;                       /* number of symbols in stbuf */

/* get a current prmap_t list (must call this before each stack trace) */
void
__fex_sym_init()
{
        struct stat statbuf;
        long n;
        int i;

        /* clear out the previous prmap_t list */
        if (pm != NULL)
                free(pm);

#endif /* ! codereview */
        pm = lpm = NULL;
        npm = 0;

        /* get the current prmap_t list */
        if (stat("/proc/self/map", &statbuf) < 0 || statbuf.st_size <= 0 ||
            (pm = (prmap_t *)malloc(statbuf.st_size)) == NULL)
                (pm = (prmap_t*)malloc(statbuf.st_size)) == NULL)
                return;

        if ((i = open("/proc/self/map", O_RDONLY)) < 0) {
        if ((i = open("/proc/self/map", O_RDONLY)) < 0)
        {
                free(pm);
                pm = NULL;
                return;
        }

#endif /* ! codereview */
        n = read(i, pm, statbuf.st_size);
        close(i);

        if (n != statbuf.st_size) {
        if (n != statbuf.st_size)
        {
                free(pm);
                pm = NULL;
        } else {
                npm = (int)(n / sizeof (prmap_t));
#endif /* ! codereview */
        }
        else
                npm = (int) (n / sizeof(prmap_t));
}

/* read ELF program headers and symbols; return -1 on error, 0 otherwise */
static int
__fex_read_syms(int fd)
{
        Elf_Ehdr h;
        Elf_Shdr *sh;
```

```
117         int i, size;

119         /* read the ELF header */
120         if (read(fd, &h, sizeof (h)) != sizeof (h))
121                 return (-1);

123         if (h.e_ident[EI_MAG0] != ELFMAG0 || h.e_ident[EI_MAG1] != ELFMAG1 ||
124             h.e_ident[EI_MAG2] != ELFMAG2 || h.e_ident[EI_MAG3] != ELFMAG3 ||
125             h.e_phentsize != sizeof (Elf_Phdr) || h.e_shentsize !=
126             sizeof (Elf_Shdr))
127                 return (-1);
 94         if (read(fd, &h, sizeof(h)) != sizeof(h))
 95                 return -1;
 96         if (h.e_ident[EI_MAG0] != ELFMAG0 ||
 97                 h.e_ident[EI_MAG1] != ELFMAG1 ||
 98                 h.e_ident[EI_MAG2] != ELFMAG2 ||
 99                 h.e_ident[EI_MAG3] != ELFMAG3 ||
100                 h.e_phentsize != sizeof(Elf_Phdr) ||
101                 h.e_shentsize != sizeof(Elf_Shdr))
102                 return -1;

129         /* get space for the program headers */
130         size = h.e_phnum * h.e_phentsize;

132         if (size > phsize) {
106         if (size > phsize)
107         {
133                 if (ph)
134                         free(ph);

136 #endif /* ! codereview */
137                 phsize = nph = 0;

139                 if ((ph = (Elf_Phdr *)malloc(size)) == NULL)
140                         return (-1);

110                 if ((ph = (Elf_Phdr*)malloc(size)) == NULL)
111                         return -1;
142                 phsize = size;
143         }

145         /* read the program headers */
146         if (lseek(fd, h.e_phoff, SEEK_SET) != h.e_phoff || read(fd, ph, size) !=
147             (ssize_t)size) {
116         if (lseek(fd, h.e_phoff, SEEK_SET) != h.e_phoff ||
117                 read(fd, ph, size) != (ssize_t)size)
118         {
148                 nph = 0;
149                 return (-1);
120                 return -1;
150         }

152 #endif /* ! codereview */
153         nph = h.e_phnum;

155         /* read the section headers */
156         size = h.e_shnum * h.e_shentsize;

158         if ((sh = (Elf_Shdr *)malloc(size)) == NULL)
159                 return (-1);

161         if (lseek(fd, h.e_shoff, SEEK_SET) != h.e_shoff || read(fd, sh, size) !=
162             (ssize_t)size) {
122         if ((sh = (Elf_Shdr*)malloc(size)) == NULL)
123                 return -1;
124         if (lseek(fd, h.e_shoff, SEEK_SET) != h.e_shoff ||
```



```
125                 read(fd, sh, size) != (ssize_t)size)
126         {
163                 free(sh);
164                 return (-1);
128                 return -1;
165         }

167         /* find the symtab section header */
168         for (i = 0; i < h.e_shnum; i++) {
132         for (i = 0; i < h.e_shnum; i++)
133         {
169                 if (sh[i].sh_type == SHT_SYMTAB)
170                         break;          /* assume there is only one */
171         }

173         if (i == h.e_shnum || sh[i].sh_size == 0 || sh[i].sh_entsize !=
174             sizeof (Elf_Sym) || sh[i].sh_link < 1 || sh[i].sh_link >=
175             h.e_shnum || sh[sh[i].sh_link].sh_type != SHT_STRTAB ||
176             sh[sh[i].sh_link].sh_size == 0) {
137         if (i == h.e_shnum || sh[i].sh_size == 0 ||
138                 sh[i].sh_entsize != sizeof(Elf_Sym) ||
139                 sh[i].sh_link < 1 || sh[i].sh_link >= h.e_shnum ||
140                 sh[sh[i].sh_link].sh_type != SHT_STRTAB ||
141                 sh[sh[i].sh_link].sh_size == 0)
142         {
177                 free(sh);
178                 return (-1);
144                 return -1;
179         }

181         /* get space for the symbol and string tables */
182         size = (int)(sh[i].sh_size + sh[sh[i].sh_link].sh_size);

184         if (size > stbufsize) {
148         size = (int) (sh[i].sh_size + sh[sh[i].sh_link].sh_size);
149         if (size > stbufsize)
150         {
185                 if (stbuf)
186                         free(stbuf);

188 #endif /* ! codereview */
189                 stbufsize = nsyms = 0;

191                 if ((stbuf = (char *)malloc(size)) == NULL) {
153                 if ((stbuf = (char*)malloc(size)) == NULL)
154                 {
192                         free(sh);
193                         return (-1);
156                         return -1;
194                 }

196 #endif /* ! codereview */
197                 stbufsize = size;
198         }

200         /* read the symbol and string tables */
201         if (lseek(fd, sh[i].sh_offset, SEEK_SET) != sh[i].sh_offset ||
202             read(fd, stbuf, sh[i].sh_size) != sh[i].sh_size ||
203             lseek(fd, sh[sh[i].sh_link].sh_offset, SEEK_SET) !=
204             sh[sh[i].sh_link].sh_offset ||
205             read(fd, stbuf + sh[i].sh_size, sh[sh[i].sh_link].sh_size) !=
206             sh[sh[i].sh_link].sh_size) {
158                         sh[sh[i].sh_link].sh_size)
159         {
207                 free(sh);
208                 return (-1);
```

```
209         }

211         nsyms = (int)(sh[i].sh_size / sh[i].sh_entsize);
212         stoffset = (int)sh[i].sh_size;
163         nsyms = (int) (sh[i].sh_size / sh[i].sh_entsize);
164         stoffset = (int) sh[i].sh_size;

214         free(sh);
215         return (0);
216 }

218 /*
219  * find the symbol corresponding to the given text address;
220  * return NULL on error, symbol address otherwise
221  */
170 /* find the symbol corresponding to the given text address;
171    return NULL on error, symbol address otherwise */
222 char *
223 __fex_sym(char *a, char **name)
224 {
225         Elf_Sym *s;
226         unsigned long fo, va, value;
227         int fd, i, j, nm;
228         char fname[PRMAPSZ + 20];
178         char                    fname[PRMAPSZ+20];

230         /* see if the last prmap_t found contains the indicated address */
231         if (lpm) {
232                 if (a >= (char *)lpm->pr_vaddr && a < (char *)lpm->pr_vaddr +
181         if (lpm)
182         {
183                 if (a >= (char*)lpm->pr_vaddr && a < (char*)lpm->pr_vaddr +
233                     lpm->pr_size)
234                         goto cont;
235         }

237         /* look for a prmap_t that contains the indicated address */
238         for (i = 0; i < npm; i++) {
239                 if (a >= (char *)pm[i].pr_vaddr && a < (char *)pm[i].pr_vaddr +
189         for (i = 0; i < npm; i++)
190         {
191                 if (a >= (char*)pm[i].pr_vaddr && a < (char*)pm[i].pr_vaddr +
240                     pm[i].pr_size)
241                         break;
242         }

244 #endif /* ! codereview */
245         if (i == npm)
246                 return (NULL);
195                 return NULL;

248         /* get an open file descriptor for the mapped object */
249         if (pm[i].pr_mapname[0] == '\0')
250                 return (NULL);

199                 return NULL;
252         strcpy(fname, "/proc/self/object/");
253         strncat(fname, pm[i].pr_mapname, PRMAPSZ);
254         fd = open(fname, O_RDONLY);

256 #endif /* ! codereview */
257         if (fd < 0)
258                 return (NULL);
203                 return NULL;

260         /* read the program headers and symbols */
```



```
261         lpm = NULL;
262         j = __fex_read_syms(fd);
263         close(fd);

265 #endif /* ! codereview */
266         if (j < 0)
267                 return (NULL);

209                 return NULL;
269         lpm = &pm[i];

271 cont:
272         /* compute the file offset corresponding to the mapped address */
273         fo = (a - (char *)lpm->pr_vaddr) + lpm->pr_offset;
214         fo = (a - (char*)lpm->pr_vaddr) + lpm->pr_offset;

275         /* find the program header containing the file offset */
276         for (i = 0; i < nph; i++) {
277                 if (ph[i].p_type == PT_LOAD && fo >= ph[i].p_offset && fo <
278                     ph[i].p_offset + ph[i].p_filesz)
217         for (i = 0; i < nph; i++)
218         {
219                 if (ph[i].p_type == PT_LOAD && fo >= ph[i].p_offset &&
220                         fo < ph[i].p_offset + ph[i].p_filesz)
279                         break;
280         }

282 #endif /* ! codereview */
283         if (i == nph)
284                 return (NULL);
223                 return NULL;

286         /* compute the virtual address corresponding to the file offset */
287         va = (fo - ph[i].p_offset) + ph[i].p_vaddr;

289         /*
290          * find the symbol in this segment with the highest value
291          * less than or equal to the virtual address
292          */
293         s = (Elf_Sym *)stbuf;
228         /* find the symbol in this segment with the highest value
229            less than or equal to the virtual address */
230         s = (Elf_Sym*)stbuf;
294         value = nm = 0;

296         for (j = 0; j < nsyms; j++) {
232         for (j = 0; j < nsyms; j++)
233         {
297                 if (s[j].st_name == 0 || s[j].st_shndx == SHN_UNDEF ||
298                     (ELF_ST_BIND(s[j].st_info) != STB_LOCAL && ELF_ST_BIND(
299                     s[j].st_info) != STB_GLOBAL && ELF_ST_BIND(s[j].st_info) !=
300                     STB_WEAK) || (ELF_ST_TYPE(s[j].st_info) != STT_NOTYPE &&
301                     ELF_ST_TYPE(s[j].st_info) != STT_OBJECT && ELF_ST_TYPE(
302                     s[j].st_info) != STT_FUNC))
235                         (ELF_ST_BIND(s[j].st_info) != STB_LOCAL &&
236                         ELF_ST_BIND(s[j].st_info) != STB_GLOBAL &&
237                         ELF_ST_BIND(s[j].st_info) != STB_WEAK) ||
238                         (ELF_ST_TYPE(s[j].st_info) != STT_NOTYPE &&
239                         ELF_ST_TYPE(s[j].st_info) != STT_OBJECT &&
240                         ELF_ST_TYPE(s[j].st_info) != STT_FUNC))
241                 {
303                         continue;
243                 }

305                 if (s[j].st_value < ph[i].p_vaddr || s[j].st_value >=
306                     ph[i].p_vaddr + ph[i].p_memsz)
```

```
                if (s[j].st_value < ph[i].p_vaddr || s[j].st_value >= ph[i].p_va
                        + ph[i].p_memsz)
                {
                        continue;
                }

                if (s[j].st_value < value || s[j].st_value > va)
                        continue;

                value = s[j].st_value;
                nm = s[j].st_name;
        }

#endif /* ! codereview */
        if (nm == 0)
                return (NULL);
                return NULL;

        /* pass back the name and return the mapped address of the symbol */
        *name = stbuf + stoffset + nm;
        fo = (value - ph[i].p_vaddr) + ph[i].p_offset;
        return ((char *)lpm->pr_vaddr + (fo - lpm->pr_offset));
        return (char*)lpm->pr_vaddr + (fo - lpm->pr_offset);
}
```

_____***unchanged_portion_omitted_***

```
**********************************************************
    1464 Thu Jun  6 20:40:07 2019
new/usr/src/lib/libm/common/m9x/fdim.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
   1 /*
   2  * CDDL HEADER START
   3  *
   4  * The contents of this file are subject to the terms of the
   5  * Common Development and Distribution License (the "License").
   6  * You may not use this file except in compliance with the License.
   7  *
   8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
   9  * or http://www.opensolaris.org/os/licensing.
  10  * See the License for the specific language governing permissions
  11  * and limitations under the License.
  12  *
  13  * When distributing Covered Code, include this CDDL HEADER in each
  14  * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  15  * If applicable, add the following below this CDDL HEADER, with the
  16  * fields enclosed by brackets "[]" replaced with your own identifying
  17  * information: Portions Copyright [yyyy] [name of copyright owner]
  18  *
  19  * CDDL HEADER END
  20  */

  22 /*
  23  * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
  24  */

  26 #endif /* ! codereview */
  27 /*
  28  * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
  29  * Use is subject to license terms.
  30  */

  32 #pragma weak fdim = __fdim

  34 /*
  35  * fdim(x,y) returns x - y if x > y, +0 if x <= y, and NaN if x and
  36  * y are unordered.
  37  *
  38  * fdim(x,y) raises overflow or inexact if x > y and x - y overflows
  39  * or is inexact.  It raises invalid if either operand is a signaling
  40  * NaN.  Otherwise, it raises no exceptions.
  41  */

  43 #include "libm.h"                       /* for islessequal macro */

  45 double
  46 __fdim(double x, double y)
  47 {
  25 __fdim(double x, double y) {
  48         if (islessequal(x, y)) {
  49                 x = 0.0;
  50                 y = -x;
  51         }

  53 #endif /* ! codereview */
  54         return (x - y);
  55 }
```

```
**********************************************************
    1526 Thu Jun  6 20:40:07 2019
new/usr/src/lib/libm/common/m9x/fdimf.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak fdimf = __fdimf

#include "libm.h"                       /* for islessequal macro */

float
__fdimf(float x, float y)
{
__fdimf(float x, float y) {
        /*
         * On SPARC v8plus/v9, this could be implemented as follows
         * (assuming %f0 = x, %f1 = y, return value left in %f0):
         *
         * fcmps        %fcc0,%f0,%f1
         * st           %g0,[scratch]   ! use fzero instead of st/ld
         * ld           [scratch],%f2   ! if VIS is available
         * fnegs        %f2,%f3
         * fmovsle      %fcc0,%f2,%f0
         * fmovsle      %fcc0,%f3,%f1
         * fsubs        %f0,%f1,%f0
         */
        if (islessequal(x, y)) {
                x = 0.0f;
                y = -x;
        }

#endif /* ! codereview */
        return (x - y);
}
```

```
**********************************************************
    1200 Thu Jun  6 20:40:08 2019
new/usr/src/lib/libm/common/m9x/fdiml.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak fdiml = __fdiml

#include "libm.h"                       /* for islessequal macro */

long double
__fdiml(long double x, long double y)
{
__fdiml(long double x, long double y) {
        if (islessequal(x, y)) {
                x = 0.0l;
                y = -x;
        }

#endif /* ! codereview */
        return (x - y);
}
```

```
**********************************************************
    3228 Thu Jun  6 20:40:08 2019
new/usr/src/lib/libm/common/m9x/feexcept.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __feclearexcept = feclearexcept
#pragma weak __feraiseexcept = feraiseexcept
#pragma weak __fetestexcept = fetestexcept
#pragma weak __fegetexceptflag = fegetexceptflag
#pragma weak __fesetexceptflag = fesetexceptflag

#pragma weak feclearexcept96 = feclearexcept
#pragma weak feraiseexcept96 = feraiseexcept
#pragma weak fetestexcept96 = fetestexcept
#pragma weak fegetexceptflag96 = fegetexceptflag
#pragma weak fesetexceptflag96 = fesetexceptflag

#include <fenv.h>
#include <sys/ieeefp.h>
#include <ucontext.h>
#include <thread.h>
#include "fex_handler.h"
#include "fenv_inlines.h"

int
feclearexcept(int e)

int feclearexcept(int e)
{
        unsigned long fsr;

        __fenv_getfsr(&fsr);
        __fenv_set_ex(fsr, __fenv_get_ex(fsr) & ~e);
        __fenv_setfsr(&fsr);
```



```
#endif /* ! codereview */
        if (fex_get_log())
                __fex_update_te();

        return (0);
        return 0;
}

/*
 *  note - __fex_hdlr depends on fetestexcept following feraiseexcept
 */
int
feraiseexcept(int e)
*  note - __fex_hdlr depends on fetestexcept following feraiseexcept
*/
int feraiseexcept(int e)
{
        volatile double t;
        unsigned long fsr;

        if (e & FE_INVALID) {
                t = 0.0;
                t /= 0.0;
        }

#endif /* ! codereview */
        if (e & FE_DIVBYZERO) {
                t = 1.0e300;
                t /= 0.0;
        }

#endif /* ! codereview */
        if (e & FE_OVERFLOW) {
                /* if overflow is not trapped, avoid raising inexact */
                __fenv_getfsr(&fsr);

#endif /* ! codereview */
                if (!(__fenv_get_te(fsr) & (1 << fp_trap_overflow))) {
                        __fenv_set_ex(fsr, __fenv_get_ex(fsr) | FE_OVERFLOW);
                        __fenv_setfsr(&fsr);
                } else {
                }
                else {
                        t = 1.0e300;
                        t *= 1.0e300;
                }
        }

#endif /* ! codereview */
        if (e & FE_UNDERFLOW) {
                /* if underflow is not trapped, avoid raising inexact */
                __fenv_getfsr(&fsr);

#endif /* ! codereview */
                if (!(__fenv_get_te(fsr) & (1 << fp_trap_underflow))) {
                        __fenv_set_ex(fsr, __fenv_get_ex(fsr) | FE_UNDERFLOW);
                        __fenv_setfsr(&fsr);
                } else {
                }
                else {
                        t = 1.0e-307;
                        t -= 1.001e-307;
                }
        }

#endif /* ! codereview */
```

```
118         if (e & FE_INEXACT) {
119                 t = 1.0e300;
120                 t += 1.0e-307;
121         }

123         return (0);
 60         return 0;
124 }

126 int
127 fetestexcept(int e)
 63 int fetestexcept(int e)
128 {
129         unsigned long fsr;

131         __fenv_getfsr(&fsr);
132         return ((int)__fenv_get_ex(fsr) & e);
 68         return (int)__fenv_get_ex(fsr) & e;
133 }

135 int
136 fegetexceptflag(fexcept_t *p, int e)
 71 int fegetexceptflag(fexcept_t *p, int e)
137 {
138         unsigned long fsr;

140         __fenv_getfsr(&fsr);
141         *p = (int)__fenv_get_ex(fsr) & e;
142         return (0);
 77         return 0;
143 }

145 int
146 fesetexceptflag(const fexcept_t *p, int e)
 80 int fesetexceptflag(const fexcept_t *p, int e)
147 {
148         unsigned long fsr;

150         __fenv_getfsr(&fsr);
151         __fenv_set_ex(fsr, (((int)__fenv_get_ex(fsr) & ~e) | (*p & e)) &
152             FE_ALL_EXCEPT);
153         __fenv_setfsr(&fsr);

155 #endif /* ! codereview */
156         if (fex_get_log())
157                 __fex_update_te();

159         return (0);
 88         return 0;
160 }
```

_____*unchanged_portion_omitted_*

```
**********************************************************
    2849 Thu Jun  6 20:40:09 2019
new/usr/src/lib/libm/common/m9x/fenv.c
11210 libm should be cstyle(10NBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __fex_merge_flags = fex_merge_flags

#pragma weak __feholdexcept = feholdexcept
#pragma weak __feupdateenv = feupdateenv
#pragma weak __fegetenv = fegetenv
#pragma weak __fesetenv = fesetenv

#pragma weak feupdateenv96 = feupdateenv
#pragma weak fegetenv96 = fegetenv
#pragma weak fesetenv96 = fesetenv

#include <fenv.h>
#include <ucontext.h>
#include <thread.h>
#include "fex_handler.h"
#include "fenv_inlines.h"

const fenv_t __fenv_dfl_env = {
        {
                { FEX_NONSTOP, (void (*)()) 0 },
                { FEX_NONSTOP, (void (*)()) 0 },
                { FEX_NONSTOP, (void (*)()) 0 },
                { FEX_NONSTOP, (void (*)()) 0 },
                { FEX_NONSTOP, (void (*)()) 0 },
                { FEX_NONSTOP, (void (*)()) 0 },
                { FEX_NONSTOP, (void (*)()) 0 },
                { FEX_NONSTOP, (void (*)()) 0 },
                { FEX_NONSTOP, (void (*)()) 0 },
                { FEX_NONSTOP, (void (*)()) 0 },
                { FEX_NONSTOP, (void (*)()) 0 },
```



```
                { FEX_NONSTOP, (void (*)()) 0 },
                { FEX_NONSTOP, (void(*)())0 },
                { FEX_NONSTOP, (void(*)())0 },
                { FEX_NONSTOP, (void(*)())0 },
                { FEX_NONSTOP, (void(*)())0 },
                { FEX_NONSTOP, (void(*)())0 },
                { FEX_NONSTOP, (void(*)())0 },
                { FEX_NONSTOP, (void(*)())0 },
                { FEX_NONSTOP, (void(*)())0 },
                { FEX_NONSTOP, (void(*)())0 },
                { FEX_NONSTOP, (void(*)())0 },
                { FEX_NONSTOP, (void(*)())0 },
                { FEX_NONSTOP, (void(*)())0 },
        },
#ifdef __x86
        0x13000000
#else
        0
#endif
};

int
feholdexcept(fenv_t *p)
int feholdexcept(fenv_t *p)
{
        (void) fegetenv(p);
        (void) feclearexcept(FE_ALL_EXCEPT);
        return (!fex_set_handling(FEX_ALL, FEX_NONSTOP, NULL));
        return !fex_set_handling(FEX_ALL, FEX_NONSTOP, NULL);
}

int
feholdexcept96(fenv_t *p)
int feholdexcept96(fenv_t *p)
{
        (void) fegetenv(p);
        (void) feclearexcept(FE_ALL_EXCEPT);
        return (fex_set_handling(FEX_ALL, FEX_NONSTOP, NULL));
        return fex_set_handling(FEX_ALL, FEX_NONSTOP, NULL);
}

int
feupdateenv(const fenv_t *p)
int feupdateenv(const fenv_t *p)
{
        unsigned long fsr;

        __fenv_getfsr(&fsr);
        (void) fesetenv(p);
        (void) feraiseexcept((int)__fenv_get_ex(fsr));
        return (0);
        return 0;
}

int
fegetenv(fenv_t *p)
int fegetenv(fenv_t *p)
{
        fex_getexcepthandler(&p->__handlers, FEX_ALL);
        __fenv_getfsr(&p->__fsr);
        return (0);
        return 0;
}

int
fesetenv(const fenv_t *p)
```

```
 76 int fesetenv(const fenv_t *p)
108 {
109         __fenv_setfsr(&p->__fsr);
110         fex_setexcepthandler(&p->__handlers, FEX_ALL);
111         return (0);
 80         return 0;
112 }

114 void
115 fex_merge_flags(const fenv_t *p)
 83 void fex_merge_flags(const fenv_t *p)
116 {
117         unsigned long fsr;

119         __fenv_getfsr(&fsr);
120         __fenv_set_ex(fsr, __fenv_get_ex(fsr) | __fenv_get_ex(p->__fsr));
121         __fenv_setfsr(&fsr);

123 #endif /* ! codereview */
124         if (fex_get_log())
125                 __fex_update_te();
126 }
```

```
**********************************************************
   12767 Thu Jun  6 20:40:09 2019
new/usr/src/lib/libm/common/m9x/fenv_inlines.h
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * This file and its contents are supplied under the terms of the
 * Common Development and Distribution License ("CDDL"), version 1.0.
 * You may only use this file in accordance with the terms of version
 * 1.0 of the CDDL.
 *
 * A full copy of the text of the CDDL should have accompanied this
 * source.  A copy of the CDDL is also available via the Internet at
 * http://www.illumos.org/license/CDDL.
 */

/*
 * Copyright 2011, Richard Lowe
 */

#ifndef _FENV_INLINES_H
#define _FENV_INLINES_H

#ifdef __GNUC__

#ifdef __cplusplus
extern "C" {
#endif

#include <sys/types.h>

#if defined(__x86)

/*
 * Floating point Control Word and Status Word
 * Definition should actually be shared with x86
 * (much of this 'amd64' code can be, in fact.)
 */
union fp_cwsw {
        uint32_t cwsw;
        struct {
                uint16_t cw;
                uint16_t sw;
        } words;
};
______unchanged_portion_omitted_

#endif

extern __GNU_INLINE void
sse_cmpeqsd(double *d1, double *d2, long long *ll1)
{
        __asm__ __volatile__(
            "cmpeqsd %2,%1\n\t"
            "movsd   %1,%0"
            : "=m" (*ll1), "+x" (*d1)
            : "x" (*d2));
}
______unchanged_portion_omitted_
#endif

#elif defined(__sparc)
extern __GNU_INLINE void
__fenv_getfsr(unsigned long *l)
{
        __asm__ __volatile__(
```



```
#if defined(__sparcv9)
            "stx %%fsr,%0\n\t"
#else
            "st  %%fsr,%0\n\t"
#endif
            : "=m" (*l));
}
______unchanged_portion_omitted_
#endif

#endif  /* __GNUC__ */

#endif /* _FENV_INLINES_H */
```

```
**********************************************************
    1460 Thu Jun  6 20:40:10 2019
new/usr/src/lib/libm/common/m9x/feprec.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __fegetprec = fegetprec
#pragma weak __fesetprec = fesetprec

#include <fenv.h>
#include <ucontext.h>
#include <thread.h>
#include "fex_handler.h"

int
fegetprec(void)
int fegetprec(void)
{
        unsigned long fsr;

        __fenv_getfsr(&fsr);
        return (__fenv_get_rp(fsr));
        return __fenv_get_rp(fsr);
}

int
fesetprec(int r)
int fesetprec(int r)
{
        unsigned long fsr;

        if (r != FE_FLTPREC && r != FE_DBLPREC && r != FE_LDBLPREC)
                return (0);

                return 0;
        __fenv_getfsr(&fsr);
```



```
        __fenv_set_rp(fsr, r);
        __fenv_setfsr(&fsr);
        return (1);
        return 1;
}
```

_____unchanged_portion_omitted_

```
**********************************************************
    1949 Thu Jun  6 20:40:10 2019
new/usr/src/lib/libm/common/m9x/feround.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __fegetround = fegetround
#pragma weak __fesetround = fesetround

#pragma weak fegetround96 = __fegetround

#include <fenv.h>
#include <ucontext.h>
#include <thread.h>
#include "fex_handler.h"
#include "fenv_inlines.h"

#if defined(__i386) && !defined(__amd64)
#include <float.h>
#endif

int
fegetround(void)
int fegetround(void)
{
        unsigned long fsr;

        __fenv_getfsr(&fsr);
        return ((int)__fenv_get_rd(fsr));
        return (int)__fenv_get_rd(fsr);
}

int
fesetround(int r)
int fesetround(int r)
{
```



```
        unsigned long fsr;

        if (r & ~3)
                return (-1);

                return -1;
        __fenv_getfsr(&fsr);
        __fenv_set_rd(fsr, r);
        __fenv_setfsr(&fsr);
#if defined(__i386) && !defined(__amd64)
        FLT_ROUNDS = (0x2D >> (r << 1)) & 3;    /* 0->1, 1->3, 2->2, 3->0 */
#endif
        return (0);
        return 0;
}

int
fesetround96(int r)
int fesetround96(int r)
{
        unsigned long fsr;

        if (r & ~3)
                return (0);

                return 0;
        __fenv_getfsr(&fsr);
        __fenv_set_rd(fsr, r);
        __fenv_setfsr(&fsr);
#if defined(__i386) && !defined(__amd64)
        FLT_ROUNDS = (0x2D >> (r << 1)) & 3;    /* 0->1, 1->3, 2->2, 3->0 */
#endif
        return (1);
        return 1;
}
_______unchanged_portion_omitted_
```

```
**********************************************************
    2426 Thu Jun  6 20:40:11 2019
new/usr/src/lib/libm/common/m9x/fex_handler.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __fex_get_handling = fex_get_handling
#pragma weak __fex_set_handling = fex_set_handling
#pragma weak __fex_getexcepthandler = fex_getexcepthandler
#pragma weak __fex_setexcepthandler = fex_setexcepthandler

#include <fenv.h>
#include <ucontext.h>
#include <thread.h>
#include "fex_handler.h"

int
fex_get_handling(int e)
int fex_get_handling(int e)
{
        struct fex_handler_data *thr_handlers;
        int i;

        thr_handlers = __fex_get_thr_handlers();

#endif /* ! codereview */
        for (i = 0; i < FEX_NUM_EXC; i++)
                if (e & (1 << i))
                        return (thr_handlers[i].__mode);

        return (FEX_NOHANDLER);
                        return thr_handlers[i].__mode;
        return FEX_NOHANDLER;
}

int
```



```
fex_set_handling(int e, int mode, void (*handler)())
int fex_set_handling(int e, int mode, void (*handler)())
{
        struct fex_handler_data *thr_handlers;
        int i;

        if (e & ~((1 << FEX_NUM_EXC) - 1))
                return (0);

                return 0;
        thr_handlers = __fex_get_thr_handlers();

#endif /* ! codereview */
        for (i = 0; i < FEX_NUM_EXC; i++) {
                if (e & (1 << i)) {
                        thr_handlers[i].__mode = mode;
                        thr_handlers[i].__handler = handler;
                }
        }

#endif /* ! codereview */
        __fex_update_te();
        return (1);
        return 1;
}

void
fex_getexcepthandler(fex_handler_t *buf, int e)
void fex_getexcepthandler(fex_handler_t *buf, int e)
{
        struct fex_handler_data *thr_handlers;
        int i;

        thr_handlers = __fex_get_thr_handlers();

#endif /* ! codereview */
        for (i = 0; i < FEX_NUM_EXC; i++)
                if (e & (1 << i))
                        (*buf)[i] = thr_handlers[i];
}

void
fex_setexcepthandler(const fex_handler_t *buf, int e)
void fex_setexcepthandler(const fex_handler_t *buf, int e)
{
        struct fex_handler_data *thr_handlers;
        int i;

        thr_handlers = __fex_get_thr_handlers();

#endif /* ! codereview */
        for (i = 0; i < FEX_NUM_EXC; i++)
                if (e & (1 << i))
                        thr_handlers[i] = (*buf)[i];

#endif /* ! codereview */
        __fex_update_te();
}
```

```
**********************************************************
    6256 Thu Jun  6 20:40:11 2019
new/usr/src/lib/libm/common/m9x/fex_handler.h
11210 libm should be cstyle(10NBLD) clean
**********************************************************
______unchanged_portion_omitted_


/* auxiliary functions in __fex_hdlr.c */
extern struct fex_handler_data *__fex_get_thr_handlers(void);
extern void __fex_update_te(void);

/* auxiliary functions in __fex_sym.c */
extern void __fex_sym_init(void);
extern char *__fex_sym(char *, char **);

/* auxiliary functions in fex_log.c */
extern void __fex_mklog(ucontext_t *, char *, int, enum fex_exception,
        int, void *);

/* system-dependent auxiliary functions */
extern enum fex_exception __fex_get_invalid_type(siginfo_t *, ucontext_t *);
extern void __fex_get_op(siginfo_t *, ucontext_t *, fex_info_t *);
extern void __fex_st_result(siginfo_t *, ucontext_t *, fex_info_t *);

/* inline templates and macros for accessing fp state */
extern void __fenv_getfsr(unsigned long *);
extern void __fenv_setfsr(const unsigned long *);

#if defined(__sparc)
#define __fenv_get_rd(X)        ((X >> 30) & 0x3)
#define __fenv_set_rd(X, Y)     X = (X & ~0xc0000000ul) | ((Y) << 30)
#endif /* ! codereview */

#define __fenv_get_te(X)        ((X >> 23) & 0x1f)
#define __fenv_set_te(X, Y)     X = (X & ~0x0f800000ul) | ((Y) << 23)
#define __fenv_get_rd(X)        ((X>>30)&0x3)
#define __fenv_set_rd(X,Y)      X=(X&~0xc0000000ul)|((Y)<<30)

#define __fenv_get_te(X)        ((X>>23)&0x1f)
#define __fenv_set_te(X,Y)      X=(X&~0x0f800000ul)|((Y)<<23)

#define __fenv_get_ex(X)        ((X>>5)&0x1f)
#define __fenv_set_ex(X,Y)      X=(X&~0x000003e0ul)|((Y)<<5)

#define __fenv_get_ex(X)        ((X >> 5) & 0x1f)
#define __fenv_set_ex(X, Y)     X = (X & ~0x000003e0ul) |((Y) << 5)
#endif /* ! codereview */
#elif defined(__x86)

extern void __fenv_getcwsw(unsigned int *);
extern void __fenv_setcwsw(const unsigned int *);

extern void __fenv_getmxcsr(unsigned int *);
extern void __fenv_setmxcsr(const unsigned int *);

#define __fenv_get_rd(X)        ((X >> 26) & 3)
#define __fenv_set_rd(X, Y)     X = (X & ~0x0c000000) | ((Y) << 26)
#define __fenv_get_rd(X)        ((X>>26)&3)
#define __fenv_set_rd(X,Y)      X=(X&~0x0c000000)|((Y)<<26)

#define __fenv_get_rp(X)        ((X >> 24) & 3)
#define __fenv_set_rp(X, Y)     X = (X & ~0x03000000) | ((Y) << 24)
#define __fenv_get_rp(X)        ((X>>24)&3)
#define __fenv_set_rp(X,Y)      X=(X&~0x03000000)|((Y)<<24)
```



```
#define __fenv_get_te(X)        ((X >> 16) & 0x3d)
#define __fenv_set_te(X, Y)     X = (X & ~0x003d0000) |((Y) << 16)
#define __fenv_get_te(X)        ((X>>16)&0x3d)
#define __fenv_set_te(X,Y)      X=(X&~0x003d0000)|((Y)<<16)

#define __fenv_get_ex(X)        (X & 0x3d)
#define __fenv_set_ex(X, Y)     X = (X & ~0x0000003d) | (Y)
#define __fenv_get_ex(X)        (X&0x3d)
#define __fenv_set_ex(X,Y)      X=(X&~0x0000003d)|(Y)

/*
 * These macros define some useful distinctions between various
 * SSE instructions.  In some cases, distinctions are made for
 * the purpose of simplifying the decoding of instructions, while
 * in other cases, they are made for the purpose of simplying the
 * emulation.  Note that these values serve as bit flags within
 * the enum values in sseinst_t.
 */
#define DOUBLE          0x100
#define SIMD            0x080
#define INTREG          0x040

typedef union {
        double          d[2];
        long long       l[2];
        float           f[4];
        int             i[4];
} sseoperand_t;
______unchanged_portion_omitted_

/* x86-specific auxiliary functions */
extern int *__fex_accrued(void);
extern void __fex_get_x86_exc(siginfo_t *, ucontext_t *);
extern int __fex_parse_sse(ucontext_t *, sseinst_t *);
extern enum fex_exception __fex_get_sse_op(ucontext_t *, sseinst_t *,
        fex_info_t *);
extern void __fex_get_simd_op(ucontext_t *, sseinst_t *,
        enum fex_exception *, fex_info_t *);
extern void __fex_st_sse_result(ucontext_t *, sseinst_t *,
        enum fex_exception, fex_info_t *);
extern void __fex_st_simd_result(ucontext_t *, sseinst_t *,
        enum fex_exception *, fex_info_t *);

#else
#error Unknown architecture
#endif

#endif  /* _M9X_FEX_HANDLER_H */
```

```
**********************************************************
    9454 Thu Jun  6 20:40:12 2019
new/usr/src/lib/libm/common/m9x/fex_log.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __fex_get_log = fex_get_log
#pragma weak __fex_set_log = fex_set_log
#pragma weak __fex_get_log_depth = fex_get_log_depth
#pragma weak __fex_set_log_depth = fex_set_log_depth
#pragma weak __fex_log_entry = fex_log_entry

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <signal.h>
#include <ucontext.h>
#include <sys/frame.h>
#include <fenv.h>
#include <sys/ieeefp.h>
#include <thread.h>
#include "fex_handler.h"

#if !defined(PC)
#if defined(REG_PC)
#define PC      REG_PC
#else
#error Neither PC nor REG_PC is defined!
#endif
#endif

static FILE *log_fp = NULL;
static mutex_t log_lock = DEFAULTMUTEX;
static int log_depth = 100;
FILE *
```



```
fex_get_log(void)

FILE *fex_get_log(void)
{
        FILE *fp;

        mutex_lock(&log_lock);
        fp = log_fp;
        mutex_unlock(&log_lock);
        return (fp);
        return fp;
}

int
fex_set_log(FILE *fp)
int fex_set_log(FILE *fp)
{
        mutex_lock(&log_lock);
        log_fp = fp;
        mutex_unlock(&log_lock);
        __fex_update_te();
        return (1);
        return 1;
}

int
fex_get_log_depth(void)
int fex_get_log_depth(void)
{
        int d;

        mutex_lock(&log_lock);
        d = log_depth;
        mutex_unlock(&log_lock);
        return (d);
        return d;
}

int
fex_set_log_depth(int d)
int fex_set_log_depth(int d)
{
        if (d < 0)
                return (0);

                return 0;
        mutex_lock(&log_lock);
        log_depth = d;
        mutex_unlock(&log_lock);
        return (1);
        return 1;
}
______unchanged_portion_omitted_

#ifdef __sparcv9
#define FRAMEP(X)               (struct frame *)((char *)(X) + (((long)(X) & \
        1) ? 2047 : 0))
#define FRAMEP(X)       (struct frame *)((char*)(X)+(((long)(X)&1)?2047:0))
#else
#define FRAMEP(X)               (struct frame *)(X)
#endif

#ifdef _LP64
#define PDIG                    "16"
#else
#define PDIG                    "8"
```

```
#endif

/*
 * look for a matching exc_list; return 1 if one is found,
 * otherwise add this one to the list and return 0
 */
static int
check_exc_list(char *addr, unsigned long code, char *stk, struct frame *fp)
/* look for a matching exc_list; return 1 if one is found,
   otherwise add this one to the list and return 0 */
static int check_exc_list(char *addr, unsigned long code, char *stk,
    struct frame *fp)
{
        struct exc_list *l, *ll = NULL;
        struct frame *f;
        int i, n;

        if (list) {
                for (l = list; l; ll = l, l = l->next) {
                        if (l->addr != addr || l->code != code)
                                continue;

#endif /* ! codereview */
                        if (log_depth < 1 || l->nstack < 1)
                                return (1);

                                return 1;
                        if (l->stack[0] != stk)
                                continue;

#endif /* ! codereview */
                        n = 1;

#endif /* ! codereview */
                        for (i = 1, f = fp; i < log_depth && i < l->nstack &&
                            f && f->fr_savpc; i++, f = FRAMEP(f->fr_savfp))
                                if (l->stack[i] != (char *)f->fr_savpc) {
                                        n = 0;
                                        break;
                                }

#endif /* ! codereview */
                        if (n)
                                return (1);
                                return 1;
                }
        }

        /* create a new exc_list structure and tack it on the list */
        for (n = 1, f = fp; n < log_depth && f && f->fr_savpc;
            n++, f = FRAMEP(f->fr_savfp))
                ;

        if ((l = (struct exc_list *)malloc(sizeof (struct exc_list) + (n - 1) *
            sizeof (char *))) != NULL) {
            n++, f = FRAMEP(f->fr_savfp)) ;
        if ((l = (struct exc_list *)malloc(sizeof(struct exc_list) +
            (n - 1) * sizeof(char *))) != NULL) {
                l->next = NULL;
                l->addr = addr;
                l->code = code;
                l->nstack = ((log_depth < 1) ? 0 : n);
                l->nstack = ((log_depth < 1)? 0 : n);
                l->stack[0] = stk;

#endif /* ! codereview */
```



```
                for (i = 1; i < n; i++) {
                        l->stack[i] = (char *)fp->fr_savpc;
                        fp = FRAMEP(fp->fr_savfp);
                }

#endif /* ! codereview */
                if (list)
                        ll->next = l;
                else
                        list = l;
        }

        return (0);
        return 0;
}

/*
 * Warning: cleverness ahead
 *
 * In the following code, the use of sprintf+write rather than fprintf
 * to send output to the log file is intentional.  The reason is that
 * fprintf is not async-signal-safe.  "But," you protest, "SIGFPE is
 * not an asynchronous signal!  It's always handled by the same thread
 * that executed the fpop that provoked it."  That's true, but a prob-
 * lem arises because (i) base conversion in fprintf can cause a fp
 * exception and (ii) my signal handler acquires a mutex lock before
 * sending output to the log file (so that outputs for entries from
 * different threads aren't interspersed).  Therefore, if the code
 * were to use fprintf, a deadlock could occur as follows:
 *
 *      Thread A                        Thread B
 *
 *      Incurs a fp exception,          Calls fprintf,
 *      acquires log_lock               acquires file rmutex lock
 *
 *      Calls fprintf,                  Incurs a fp exception,
 *      waits for file rmutex lock      waits for log_lock
 *
 * (I could just verify that fprintf doesn't hold the rmutex lock while
 * it's doing the base conversion, but since efficiency is of little
 * concern here, I opted for the safe and dumb route.)
 */
static void
print_stack(int fd, char *addr, struct frame *fp)
* Warning: cleverness ahead
*
* In the following code, the use of sprintf+write rather than fprintf
* to send output to the log file is intentional.  The reason is that
* fprintf is not async-signal-safe.  "But," you protest, "SIGFPE is
* not an asynchronous signal!  It's always handled by the same thread
* that executed the fpop that provoked it."  That's true, but a prob-
* lem arises because (i) base conversion in fprintf can cause a fp
* exception and (ii) my signal handler acquires a mutex lock before
* sending output to the log file (so that outputs for entries from
* different threads aren't interspersed).  Therefore, if the code
* were to use fprintf, a deadlock could occur as follows:
*
*       Thread A                        Thread B
*
*       Incurs a fp exception,          Calls fprintf,
*       acquires log_lock               acquires file rmutex lock
*
*       Calls fprintf,                  Incurs a fp exception,
*       waits for file rmutex lock      waits for log_lock
*
* (I could just verify that fprintf doesn't hold the rmutex lock while
```

```
141 * it's doing the base conversion, but since efficiency is of little
142 * concern here, I opted for the safe and dumb route.)
143 */

145 static void print_stack(int fd, char *addr, struct frame *fp)
223 {
224         int i;
225         char *name, buf[30];

227         for (i = 0; i < log_depth && addr != NULL; i++) {
228                 if (__fex_sym(addr, &name) != NULL) {
229                         write(fd, buf, sprintf(buf, "  0x%0" PDIG "lx  ",
230                             (long)addr));
231                         write(fd, name, strlen(name));
232                         write(fd, "\n", 1);

234                         if (strcmp(name, "main") == 0)
156                         if (!strcmp(name, "main"))
235                                 break;
236                 } else {
237                         write(fd, buf, sprintf(buf, "  0x%0" PDIG "lx\n",
238                             (long)addr));
239                 }

241 #endif /* ! codereview */
242                 if (fp == NULL)
243                         break;

245 #endif /* ! codereview */
246                 addr = (char *)fp->fr_savpc;
247                 fp = FRAMEP(fp->fr_savfp);
248         }
249 }

251 void
252 fex_log_entry(const char *msg)
162 void fex_log_entry(const char *msg)
253 {
254         ucontext_t uc;
255         struct frame *fp;
256         char *stk;
257         int fd;

259         /* if logging is disabled, just return */
260         mutex_lock(&log_lock);

262 #endif /* ! codereview */
263         if (log_fp == NULL) {
264                 mutex_unlock(&log_lock);
265                 return;
266         }

268         /*
269          * get the frame pointer from the current context and
270          * pop our own frame
271          */
171         /* get the frame pointer from the current context and
172            pop our own frame */
272         getcontext(&uc);
273 #if defined(__sparc) || defined(__amd64)
274         fp = FRAMEP(uc.uc_mcontext.gregs[REG_SP]);
275 #elif defined(__i386)                   /* !defined(__amd64) */
276         fp = FRAMEP(uc.uc_mcontext.gregs[EBP]);
277 #else
278 #error Unknown architecture
279 #endif
```



```
281 #endif /* ! codereview */
282         if (fp == NULL) {
283                 mutex_unlock(&log_lock);
284                 return;
285         }

287 #endif /* ! codereview */
288         stk = (char *)fp->fr_savpc;
289         fp = FRAMEP(fp->fr_savfp);

291         /* if we've already logged this message here, don't make an entry */
292         if (check_exc_list(stk, (unsigned long)msg, stk, fp)) {
293                 mutex_unlock(&log_lock);
294                 return;
295         }

297         /* make an entry */
298         fd = fileno(log_fp);
299         write(fd, "fex_log_entry: ", 15);
300         write(fd, msg, strlen(msg));
301         write(fd, "\n", 1);
302         __fex_sym_init();
303         print_stack(fd, stk, fp);
304         mutex_unlock(&log_lock);
305 }

307 static const char *exception[FEX_NUM_EXC] = {
308         "inexact result", "division by zero", "underflow", "overflow",
309         "invalid operation (0/0)", "invalid operation (inf/inf)",
310         "invalid operation (inf-inf)", "invalid operation (0*inf)",
311         "invalid operation (sqrt)", "invalid operation (snan)",
312         "invalid operation (int)", "invalid operation (cmp)"
181         "inexact result",
182         "division by zero",
183         "underflow",
184         "overflow",
185         "invalid operation (0/0)",
186         "invalid operation (inf/inf)",
187         "invalid operation (inf-inf)",
188         "invalid operation (0*inf)",
189         "invalid operation (sqrt)",
190         "invalid operation (snan)",
191         "invalid operation (int)",
192         "invalid operation (cmp)"
313 };

315 void
316 __fex_mklog(ucontext_t *uap, char *addr, int f, enum fex_exception e, int m,
317     void *p)
196 __fex_mklog(ucontext_t *uap, char *addr, int f, enum fex_exception e,
197     int m, void *p)
318 {
319         struct  frame *fp;
320         char *stk, *name, buf[30];
321         int fd;

323         /* if logging is disabled, just return */
324         mutex_lock(&log_lock);

326 #endif /* ! codereview */
327         if (log_fp == NULL) {
328                 mutex_unlock(&log_lock);
329                 return;
330         }
```

```
332         /* get stack info */
333 #if defined(__sparc)
334         stk = (char *)uap->uc_mcontext.gregs[REG_PC];
205         stk = (char*)uap->uc_mcontext.gregs[REG_PC];
335         fp = FRAMEP(uap->uc_mcontext.gregs[REG_SP]);
336 #elif defined(__amd64)
337         stk = (char *)uap->uc_mcontext.gregs[REG_PC];
208         stk = (char*)uap->uc_mcontext.gregs[REG_PC];
338         fp = FRAMEP(uap->uc_mcontext.gregs[REG_RBP]);
339 #elif defined(__i386)                   /* !defined(__amd64) */
340         stk = (char *)uap->uc_mcontext.gregs[PC];
211         stk = (char*)uap->uc_mcontext.gregs[PC];
341         fp = FRAMEP(uap->uc_mcontext.gregs[EBP]);
342 #else
343 #error Unknown architecture
344 #endif

346         /*
347          * if the handling mode is the default and this exception's
348          * flag is already raised, don't make an entry
349          */
217         /* if the handling mode is the default and this exception's
218            flag is already raised, don't make an entry */
350         if (m == FEX_NONSTOP) {
351                 switch (e) {
352                 case fex_inexact:

354 #endif /* ! codereview */
355                         if (f & FE_INEXACT) {
356                                 mutex_unlock(&log_lock);
357                                 return;
358                         }

360 #endif /* ! codereview */
361                         break;
362                 case fex_underflow:

364 #endif /* ! codereview */
365                         if (f & FE_UNDERFLOW) {
366                                 mutex_unlock(&log_lock);
367                                 return;
368                         }

370 #endif /* ! codereview */
371                         break;
372                 case fex_overflow:

374 #endif /* ! codereview */
375                         if (f & FE_OVERFLOW) {
376                                 mutex_unlock(&log_lock);
377                                 return;
378                         }

380 #endif /* ! codereview */
381                         break;
382                 case fex_division:

384 #endif /* ! codereview */
385                         if (f & FE_DIVBYZERO) {
386                                 mutex_unlock(&log_lock);
387                                 return;
388                         }

390 #endif /* ! codereview */
391                         break;
392                 default:
```



```
394 #endif /* ! codereview */
395                         if (f & FE_INVALID) {
396                                 mutex_unlock(&log_lock);
397                                 return;
398                         }

400 #endif /* ! codereview */
401                         break;
402                 }
403         }

405         /*
406          * if we've already logged this exception at this address,
407          * don't make an entry
408          */
222         /* if we've already logged this exception at this address,
223            don't make an entry */
409         if (check_exc_list(addr, (unsigned long)e, stk, fp)) {
410                 mutex_unlock(&log_lock);
411                 return;
412         }

414         /* make an entry */
415         fd = fileno(log_fp);
416         write(fd, "Floating point ", 15);
417         write(fd, exception[e], strlen(exception[e]));
418         write(fd, buf, sprintf(buf, " at 0x%0" PDIG "lx", (long)addr));
419         __fex_sym_init();

421 #endif /* ! codereview */
422         if (__fex_sym(addr, &name) != NULL) {
423                 write(fd, " ", 1);
424                 write(fd, name, strlen(name));
425         }

427 #endif /* ! codereview */
428         switch (m) {
429         case FEX_NONSTOP:
430                 write(fd, ", nonstop mode\n", 15);
431                 break;

433         case FEX_ABORT:
434                 write(fd, ", abort\n", 8);
435                 break;

437         case FEX_NOHANDLER:

439 #endif /* ! codereview */
440                 if (p == (void *)SIG_DFL) {
441                         write(fd, ", handler: SIG_DFL\n", 19);
442                         break;
443                 } else if (p == (void *)SIG_IGN) {
235                 }
236                 else if (p == (void *)SIG_IGN) {
444                         write(fd, ", handler: SIG_IGN\n", 19);
445                         break;
446                 }

448         /* FALLTHROUGH */
240                 /* fall through*/
449         default:
450                 write(fd, ", handler: ", 11);

452 #endif /* ! codereview */
453                 if (__fex_sym((char *)p, &name) != NULL) {
```

```
                        write(fd, name, strlen(name));
                        write(fd, "\n", 1);
                } else {
                        write(fd, buf, sprintf(buf, "0x%0" PDIG "lx\n",
                            (long)p));
                }

#endif /* ! codereview */
                break;
        }

#endif /* ! codereview */
        print_stack(fd, stk, fp);
        mutex_unlock(&log_lock);
}
```

```
**********************************************************
   11380 Thu Jun  6 20:40:12 2019
new/usr/src/lib/libm/common/m9x/fma.c
11210 libm should be cstyle(10NBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak fma = __fma

#include "libm.h"
#include "fma.h"
#include "fenv_inlines.h"

#if defined(__sparc)

static const union {
        unsigned i[2];
        double d;
} C[] = {
______unchanged_portion_omitted_

#define half            C[0].d
#define two             C[1].d
#define two52           C[2].d
#define two27           C[3].d
#define twom26          C[4].d
#define twom32          C[5].d
#define twom64          C[6].d
#define huge            C[7].d
#define tiny            C[8].d
#define tiny2           C[9].d

static const unsigned int fsr_rm = 0xc0000000u;

/*
 * fma for SPARC: 64-bit double precision, big-endian
 */
```



```
double
__fma(double x, double y, double z)
{
__fma(double x, double y, double z) {
        union {
                unsigned i[2];
                double d;
        } xx, yy, zz;

#endif /* ! codereview */
        double xhi, yhi, xlo, ylo, t;
        unsigned int xy0, xy1, xy2, xy3, z0, z1, z2, z3, fsr, rm, sticky;
        int hx, hy, hz, ex, ey, ez, exy, sxy, sz, e, ibit;
        volatile double dummy;

        /* extract the high order words of the arguments */
        xx.d = x;
        yy.d = y;
        zz.d = z;
        hx = xx.i[0] & ~0x80000000;
        hy = yy.i[0] & ~0x80000000;
        hz = zz.i[0] & ~0x80000000;

        /* dispense with inf, nan, and zero cases */
        if (hx >= 0x7ff00000 || hy >= 0x7ff00000 || (hx | xx.i[1]) == 0 || (hy |
            yy.i[1]) == 0)              /* x or y is inf, nan, or zero */
        if (hx >= 0x7ff00000 || hy >= 0x7ff00000 || (hx | xx.i[1]) == 0 ||
                (hy | yy.i[1]) == 0)    /* x or y is inf, nan, or zero */
                return (x * y + z);

        if (hz >= 0x7ff00000)   /* z is inf or nan */
                return (x + z); /* avoid spurious under/overflow in x * y */

        if ((hz | zz.i[1]) == 0) /* z is zero */
                /*
                 * x * y isn't zero but could underflow to zero,
                 * so don't add z, lest we perturb the sign
                 */
                return (x * y);

        /*
         * now x, y, and z are all finite and nonzero; save the fsr and
         * set round-to-negative-infinity mode (and clear nonstandard
         * mode before we try to scale subnormal operands)
         */
        __fenv_getfsr32(&fsr);
        __fenv_setfsr32(&fsr_rm);

        /* extract signs and exponents, and normalize subnormals */
        sxy = (xx.i[0] ^ yy.i[0]) & 0x80000000;
        sz = zz.i[0] & 0x80000000;
        ex = hx >> 20;

#endif /* ! codereview */
        if (!ex) {
                xx.d = x * two52;
                ex = ((xx.i[0] & ~0x80000000) >> 20) - 52;
        }

#endif /* ! codereview */
        ey = hy >> 20;

#endif /* ! codereview */
        if (!ey) {
                yy.d = y * two52;
                ey = ((yy.i[0] & ~0x80000000) >> 20) - 52;
```

```
        }

#endif /* ! codereview */
        ez = hz >> 20;

#endif /* ! codereview */
        if (!ez) {
                zz.d = z * two52;
                ez = ((zz.i[0] & ~0x80000000) >> 20) - 52;
        }

        /* multiply x*y to 106 bits */
        exy = ex + ey - 0x3ff;
        xx.i[0] = (xx.i[0] & 0xfffff) | 0x3ff00000;
        yy.i[0] = (yy.i[0] & 0xfffff) | 0x3ff00000;
        x = xx.d;
        y = yy.d;
        xhi = ((x + twom26) + two27) - two27;
        yhi = ((y + twom26) + two27) - two27;
        xlo = x - xhi;
        ylo = y - yhi;
        x *= y;
        y = ((xhi * yhi - x) + xhi * ylo + xlo * yhi) + xlo * ylo;

#endif /* ! codereview */
        if (x >= two) {
                x *= half;
                y *= half;
                exy++;
        }

        /* extract the significands */
        xx.d = x;
        xy0 = (xx.i[0] & 0xfffff) | 0x100000;
        xy1 = xx.i[1];
        yy.d = t = y + twom32;
        xy2 = yy.i[1];
        yy.d = (y - (t - twom32)) + twom64;
        xy3 = yy.i[1];
        z0 = (zz.i[0] & 0xfffff) | 0x100000;
        z1 = zz.i[1];
        z2 = z3 = 0;

        /*
         * now x*y is represented by sxy, exy, and xy[0-3], and z is
         * represented likewise; swap if need be so |xy| <= |z|
         */
        if (exy > ez || (exy == ez && (xy0 > z0 || (xy0 == z0 && (xy1 > z1 ||
            (xy1 == z1 && (xy2 | xy3) != 0)))))) {
                e = sxy;
                sxy = sz;
                sz = e;
                e = exy;
                exy = ez;
                ez = e;
                e = xy0;
                xy0 = z0;
                z0 = e;
                e = xy1;
                xy1 = z1;
                z1 = e;
                z2 = xy2;
                xy2 = 0;
                z3 = xy3;
                xy3 = 0;
        if (exy > ez || (exy == ez && (xy0 > z0 || (xy0 == z0 &&
```



```
                (xy1 > z1 || (xy1 == z1 && (xy2 | xy3) != 0)))))) {
                e = sxy; sxy = sz; sz = e;
                e = exy; exy = ez; ez = e;
                e = xy0; xy0 = z0; z0 = e;
                e = xy1; xy1 = z1; z1 = e;
                z2 = xy2; xy2 = 0;
                z3 = xy3; xy3 = 0;
        }

        /* shift the significand of xy keeping a sticky bit */
        e = ez - exy;

#endif /* ! codereview */
        if (e > 116) {
                xy0 = xy1 = xy2 = 0;
                xy3 = 1;
        } else if (e >= 96) {
                sticky = xy3 | xy2 | xy1 | ((xy0 << 1) << (127 - e));
                xy3 = xy0 >> (e - 96);

#endif /* ! codereview */
                if (sticky)
                        xy3 |= 1;

#endif /* ! codereview */
                xy0 = xy1 = xy2 = 0;
        } else if (e >= 64) {
                sticky = xy3 | xy2 | ((xy1 << 1) << (95 - e));
                xy3 = (xy1 >> (e - 64)) | ((xy0 << 1) << (95 - e));

#endif /* ! codereview */
                if (sticky)
                        xy3 |= 1;

#endif /* ! codereview */
                xy2 = xy0 >> (e - 64);
                xy0 = xy1 = 0;
        } else if (e >= 32) {
                sticky = xy3 | ((xy2 << 1) << (63 - e));
                xy3 = (xy2 >> (e - 32)) | ((xy1 << 1) << (63 - e));

#endif /* ! codereview */
                if (sticky)
                        xy3 |= 1;

#endif /* ! codereview */
                xy2 = (xy1 >> (e - 32)) | ((xy0 << 1) << (63 - e));
                xy1 = xy0 >> (e - 32);
                xy0 = 0;
        } else if (e) {
                sticky = (xy3 << 1) << (31 - e);
                xy3 = (xy3 >> e) | ((xy2 << 1) << (31 - e));

#endif /* ! codereview */
                if (sticky)
                        xy3 |= 1;

#endif /* ! codereview */
                xy2 = (xy2 >> e) | ((xy1 << 1) << (31 - e));
                xy1 = (xy1 >> e) | ((xy0 << 1) << (31 - e));
                xy0 >>= e;
        }

        /* if this is a magnitude subtract, negate the significand of xy */
        if (sxy ^ sz) {
                xy0 = ~xy0;
```

```
                xy1 = ~xy1;
                xy2 = ~xy2;
                xy3 = -xy3;

#endif /* ! codereview */
                if (xy3 == 0)
                        if (++xy2 == 0)
                                if (++xy1 == 0)
                                        xy0++;
        }

        /* add, propagating carries */
        z3 += xy3;
        e = (z3 < xy3);
        z2 += xy2;

#endif /* ! codereview */
        if (e) {
                z2++;
                e = (z2 <= xy2);
        } else {
        } else
                e = (z2 < xy2);
        }

#endif /* ! codereview */
        z1 += xy1;

#endif /* ! codereview */
        if (e) {
                z1++;
                e = (z1 <= xy1);
        } else {
        } else
                e = (z1 < xy1);
        }

#endif /* ! codereview */
        z0 += xy0;

#endif /* ! codereview */
        if (e)
                z0++;

        /* postnormalize and collect rounding information into z2 */
        if (ez < 1) {
                /* result is tiny; shift right until exponent is within range */
                e = 1 - ez;

#endif /* ! codereview */
                if (e > 56) {
                        z2 = 1;         /* result can't be exactly zero */
                        z0 = z1 = 0;
                } else if (e >= 32) {
                        sticky = z3 | z2 | ((z1 << 1) << (63 - e));
                        z2 = (z1 >> (e - 32)) | ((z0 << 1) << (63 - e));

#endif /* ! codereview */
                        if (sticky)
                                z2 |= 1;

#endif /* ! codereview */
                        z1 = z0 >> (e - 32);
                        z0 = 0;
                } else {
                        sticky = z3 | (z2 << 1) << (31 - e);
```



```
                        z2 = (z2 >> e) | ((z1 << 1) << (31 - e));

#endif /* ! codereview */
                        if (sticky)
                                z2 |= 1;

#endif /* ! codereview */
                        z1 = (z1 >> e) | ((z0 << 1) << (31 - e));
                        z0 >>= e;
                }

#endif /* ! codereview */
                ez = 1;
        } else if (z0 >= 0x200000) {
                /* carry out; shift right by one */
                sticky = (z2 & 1) | z3;
                z2 = (z2 >> 1) | (z1 << 31);

#endif /* ! codereview */
                if (sticky)
                        z2 |= 1;

#endif /* ! codereview */
                z1 = (z1 >> 1) | (z0 << 31);
                z0 >>= 1;
                ez++;
        } else {
                if (z0 < 0x100000 && (z0 | z1 | z2 | z3) != 0) {
                        /*
                         * borrow/cancellation; shift left as much as
                         * exponent allows
                         */
                        while (!(z0 | (z1 & 0xffe00000)) && ez >= 33) {
                                z0 = z1;
                                z1 = z2;
                                z2 = z3;
                                z3 = 0;
                                ez -= 32;
                        }

#endif /* ! codereview */
                        while (z0 < 0x100000 && ez > 1) {
                                z0 = (z0 << 1) | (z1 >> 31);
                                z1 = (z1 << 1) | (z2 >> 31);
                                z2 = (z2 << 1) | (z3 >> 31);
                                z3 <<= 1;
                                ez--;
                        }
                }

#endif /* ! codereview */
                if (z3)
                        z2 |= 1;
        }

        /* get the rounding mode and clear current exceptions */
        rm = fsr >> 30;
        fsr &= ~FSR_CEXC;

        /* strip off the integer bit, if there is one */
        ibit = z0 & 0x100000;

        if (ibit) {
        if (ibit)
                z0 -= 0x100000;
        } else {
```

```
        else {
                ez = 0;

#endif /* ! codereview */
                if (!(z0 | z1 | z2)) {  /* exact zero */
                        zz.i[0] = rm == FSR_RM ? 0x80000000 : 0;
                        zz.i[1] = 0;
                        __fenv_setfsr32(&fsr);
                        return (zz.d);
                }
        }

        /*
         * flip the sense of directed roundings if the result is negative;
         * the logic below applies to a positive result
         */
        if (sz)
                rm ^= rm >> 1;

        /* round and raise exceptions */
        if (z2) {
                fsr |= FSR_NXC;

                /* decide whether to round the fraction up */
                if (rm == FSR_RP || (rm == FSR_RN && (z2 > 0x80000000u || (z2 ==
                    0x80000000u && (z1 & 1))))) {
                if (rm == FSR_RP || (rm == FSR_RN && (z2 > 0x80000000u ||
                        (z2 == 0x80000000u && (z1 & 1))))) {
                        /* round up and renormalize if necessary */
                        if (++z1 == 0) {
                                if (++z0 == 0x100000) {
                                        z0 = 0;
                                        ez++;
                                }
                        }
                }
        }

        /* check for under/overflow */
        if (ez >= 0x7ff) {
                if (rm == FSR_RN || rm == FSR_RP) {
                        zz.i[0] = sz | 0x7ff00000;
                        zz.i[1] = 0;
                } else {
                        zz.i[0] = sz | 0x7effffff;
                        zz.i[1] = 0xffffffff;
                }

#endif /* ! codereview */
                fsr |= FSR_OFC | FSR_NXC;
        } else {
                zz.i[0] = sz | (ez << 20) | z0;
                zz.i[1] = z1;

                /*
                 * !ibit => exact result was tiny before rounding,
                 * z2 nonzero => result delivered is inexact
                 */
                if (!ibit) {
                        if (z2)
                                fsr |= FSR_UFC | FSR_NXC;
                        else if (fsr & FSR_UFM)
                                fsr |= FSR_UFC;
                }
        }
```



```
        /* restore the fsr and emulate exceptions as needed */
        if ((fsr & FSR_CEXC) & (fsr >> 23)) {
                __fenv_setfsr32(&fsr);

#endif /* ! codereview */
                if (fsr & FSR_OFC) {
                        dummy = huge;
                        dummy *= huge;
                } else if (fsr & FSR_UFC) {
                        dummy = tiny;

#endif /* ! codereview */
                        if (fsr & FSR_NXC)
                                dummy *= tiny;
                        else
                                dummy -= tiny2;
                } else {
                        dummy = huge;
                        dummy += tiny;
                }
        } else {
                fsr |= (fsr & 0x1f) << 5;
                __fenv_setfsr32(&fsr);
        }

#endif /* ! codereview */
        return (zz.d);
}

#elif defined(__x86)

#if defined(__amd64)
#define NI              4
#else
#define NI              3
#endif

/*
 *  fma for x86: 64-bit double precision, little-endian
 */
double
__fma(double x, double y, double z)
{
__fma(double x, double y, double z) {
        union {
                unsigned i[NI];
                long double e;
        } xx, yy, zz;

#endif /* ! codereview */
        long double xe, ye, xhi, xlo, yhi, ylo;
        int ex, ey, ez;
        unsigned cwsw, oldcwsw, rm;

        /* convert the operands to double extended */
        xx.e = (long double)x;
        yy.e = (long double)y;
        zz.e = (long double)z;
        xx.e = (long double) x;
        yy.e = (long double) y;
        zz.e = (long double) z;

        /* extract the exponents of the arguments */
        ex = xx.i[2] & 0x7fff;
        ey = yy.i[2] & 0x7fff;
        ez = zz.i[2] & 0x7fff;
```

```
        /* dispense with inf, nan, and zero cases */
        if (ex == 0x7fff || ey == 0x7fff || ex == 0 || ey == 0)
                /* x or y is inf, nan, or zero */
                return ((double)(xx.e * yy.e + zz.e));
                return ((double) (xx.e * yy.e + zz.e));

        if (ez >= 0x7fff)               /* z is inf or nan */
                return ((double)(xx.e + zz.e));

                return ((double) (xx.e + zz.e));
        /* avoid spurious inexact in x * y */

        /*
         * save the control and status words, mask all exceptions, and
         * set rounding to 64-bit precision and to-nearest
         */
        __fenv_getcwsw(&oldcwsw);
        cwsw = (oldcwsw & 0xf0c0ffff) | 0x033f0000;
        __fenv_setcwsw(&cwsw);

        /* multiply x*y to 106 bits */
        xe = xx.e;
        xx.i[0] = 0;
        xhi = xx.e;                     /* hi 32 bits */
        xlo = xe - xhi;                 /* lo 21 bits */
        ye = yy.e;
        yy.i[0] = 0;
        yhi = yy.e;
        ylo = ye - yhi;
        xe = xe * ye;
        ye = ((xhi * yhi - xe) + xhi * ylo + xlo * yhi) + xlo * ylo;

        /* distill the sum of xe, ye, and z */
        xhi = ye + zz.e;
        yhi = xhi - ye;
        xlo = (zz.e - yhi) + (ye - (xhi - yhi));
        /* now (xhi,xlo) = ye + z */

        yhi = xe + xhi;
        ye = yhi - xe;
        ylo = (xhi - ye) + (xe - (yhi - ye));   /* now (yhi,ylo) = xe + xhi */

        xhi = xlo + ylo;
        xe = xhi - xlo;
        xlo = (ylo - xe) + (xlo - (xhi - xe));  /* now (xhi,xlo) = xlo + ylo */

        yy.e = yhi + xhi;
        ylo = (yhi - yy.e) + xhi;               /* now (yy.e,ylo) = xhi + yhi */

        if (yy.i[1] != 0) {                     /* yy.e is nonzero */
                /* perturb yy.e if its least significant 10 bits are zero */
                if (!(yy.i[0] & 0x3ff)) {
                        xx.e = ylo + xlo;

#endif /* ! codereview */
                        if (xx.i[1] != 0) {
                                xx.i[2] = (xx.i[2] & 0x8000) | ((yy.i[2] &
                                    0x7fff) - 63);
                                xx.i[2] = (xx.i[2] & 0x8000) |
                                        ((yy.i[2] & 0x7fff) - 63);
                                xx.i[1] = 0x80000000;
                                xx.i[0] = 0;
                                yy.e += xx.e;
                        }
                }
```



```
        } else {
                /* set sign of zero result according to rounding direction */
                rm = oldcwsw & 0x0c000000;
                yy.i[2] = ((rm == FCW_RM) ? 0x8000 : 0);
                yy.i[2] = ((rm == FCW_RM)? 0x8000 : 0);
        }

        /*
         * restore the control and status words and convert the result
         * to double
         */
        __fenv_setcwsw(&oldcwsw);
        return ((double)yy.e);
        return ((double) yy.e);
}

#else
#error Unknown architecture
#endif
```

```
**********************************************************
    2813 Thu Jun  6 20:40:13 2019
new/usr/src/lib/libm/common/m9x/fma.h
11210 libm should be cstyle(10NBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#ifndef _FMA_H
#define _FMA_H

#ifdef __cplusplus
extern "C" {
#endif

#ifdef __sparc

/*
 * Common definitions for fma routines (SPARC)
 */

/* fsr fields */

/* current exception bits */
#define FSR_NXC         0x1
#define FSR_DZC         0x2
#define FSR_UFC         0x4
#define FSR_OFC         0x8
#define FSR_NVC         0x10
#define FSR_CEXC        0x1f            /* mask for all cexc bits */

/* accrued exception bits */
#define FSR_NXA         0x20
#define FSR_DZA         0x40
#define FSR_UFA         0x80
#define FSR_OFA         0x100
#define FSR_NVA         0x200
```



```
/* trap enable bits */
#define FSR_NXM         0x00800000
#define FSR_DZM         0x01000000
#define FSR_UFM         0x02000000
#define FSR_OFM         0x04000000
#define FSR_NVM         0x08000000

/* rounding directions (right-adjusted) */
#define FSR_RN          0
#define FSR_RZ          1
#define FSR_RP          2
#define FSR_RM          3

/* inline templates */
extern void __fenv_getfsr32(unsigned int *);
extern void __fenv_setfsr32(const unsigned int *);

#endif /* __sparc */


#if defined(__x86)

/*
 * Common definitions for fma routines (x86)
 */

/* control and status word fields */

/* exception flags */
#define FSW_NV          0x1
#define FSW_DN          0x2
#define FSW_DZ          0x4
#define FSW_OF          0x8
#define FSW_UF          0x10
#define FSW_NX          0x20

/* exception masks */
#define FCW_NVM         0x00010000
#define FCW_DNM         0x00020000
#define FCW_DZM         0x00040000
#define FCW_OFM         0x00080000
#define FCW_UFM         0x00100000
#define FCW_NXM         0x00200000
#define FCW_ALLM        0x003f0000

/* rounding directions */
#define FCW_RN          0x00000000
#define FCW_RM          0x04000000
#define FCW_RP          0x08000000
#define FCW_RZ          0x0c000000

/* rounding precisions */
#define FCW_P24         0x00000000
#define FCW_P53         0x02000000
#define FCW_P64         0x03000000

/* inline templates */
extern void __fenv_getcwsw(unsigned int *);
extern void __fenv_setcwsw(const unsigned int *);

#endif /* __x86 */

#ifdef __cplusplus
}
#endif
```

```
#endif /* _FMA_H */
```

```
**********************************************************
    5641 Thu Jun  6 20:40:13 2019
new/usr/src/lib/libm/common/m9x/fmaf.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak fmaf = __fmaf

#include "libm.h"
#include "fma.h"
#include "fenv_inlines.h"

#if defined(__sparc)

/*
 * fmaf for SPARC: 32-bit single precision, big-endian
 */
float
__fmaf(float x, float y, float z)
{
__fmaf(float x, float y, float z) {
        union {
                unsigned i[2];
                double d;
        } xy, zz;

#endif /* ! codereview */
        unsigned u, s;
        int exy, ez;

        /*
         * the following operations can only raise the invalid exception,
         * and then only if either x*y is of the form Inf*0 or one of x,
         * y, or z is a signaling NaN
         */
        xy.d = (double)x * y;
```



```
        zz.d = (double)z;
        xy.d = (double) x * y;
        zz.d = (double) z;

        /*
         * if the sum xy + z will be exact, just compute it and cast the
         * result to float
         */
        exy = (xy.i[0] >> 20) & 0x7ff;
        ez = (zz.i[0] >> 20) & 0x7ff;

#endif /* ! codereview */
        if ((ez - exy <= 4 && exy - ez <= 28) || exy == 0x7ff || exy == 0 ||
            ez == 0x7ff || ez == 0)
                return ((float)(xy.d + zz.d));
                ez == 0x7ff || ez == 0) {
                return ((float) (xy.d + zz.d));
        }

        /*
         * collapse the tail of the smaller summand into a "sticky bit"
         * so that the sum can be computed without error
         */
        if (ez > exy) {
                if (ez - exy < 31) {
                        u = xy.i[1];
                        s = 2 << (ez - exy);

#endif /* ! codereview */
                        if (u & (s - 1))
                                u |= s;

#endif /* ! codereview */
                        xy.i[1] = u & ~(s - 1);
                } else if (ez - exy < 51) {
                        u = xy.i[0];
                        s = 1 << (ez - exy - 31);

#endif /* ! codereview */
                        if ((u & (s - 1)) | xy.i[1])
                                u |= s;

#endif /* ! codereview */
                        xy.i[0] = u & ~(s - 1);
                        xy.i[1] = 0;
                } else {
                        /* collapse all of xy into a single bit */
                        xy.i[0] = (xy.i[0] & 0x80000000) | ((ez - 51) << 20);
                        xy.i[1] = 0;
                }
        } else {
                if (exy - ez < 31) {
                        u = zz.i[1];
                        s = 2 << (exy - ez);

#endif /* ! codereview */
                        if (u & (s - 1))
                                u |= s;

#endif /* ! codereview */
                        zz.i[1] = u & ~(s - 1);
                } else if (exy - ez < 51) {
                        u = zz.i[0];
                        s = 1 << (exy - ez - 31);

#endif /* ! codereview */
```

```
                        if ((u & (s - 1)) | zz.i[1])
                                u |= s;

#endif /* ! codereview */
                        zz.i[0] = u & ~(s - 1);
                        zz.i[1] = 0;
                } else {
                        /* collapse all of zz into a single bit */
                        zz.i[0] = (zz.i[0] & 0x80000000) | ((exy - 51) << 20);
                        zz.i[1] = 0;
                }
        }

        return ((float)(xy.d + zz.d));
        return ((float) (xy.d + zz.d));
}

#elif defined(__x86)

#if defined(__amd64)
#define NI              4
#else
#define NI              3
#endif

/*
 * fmaf for x86: 32-bit single precision, little-endian
 */
float
__fmaf(float x, float y, float z)
{
__fmaf(float x, float y, float z) {
        union {
                unsigned i[NI];
                long double e;
        } xy, zz;

#endif /* ! codereview */
        unsigned u, s, cwsw, oldcwsw;
        int exy, ez;

        /* set rounding precision to 64 bits */
        __fenv_getcwsw(&oldcwsw);
        cwsw = (oldcwsw & 0xfcffffff) | 0x03000000;
        __fenv_setcwsw(&cwsw);

        /*
         * the following operations can only raise the invalid exception,
         * and then only if either x*y is of the form Inf*0 or one of x,
         * y, or z is a signaling NaN
         */
        xy.e = (long double)x * y;
        zz.e = (long double)z;
        xy.e = (long double) x * y;
        zz.e = (long double) z;

        /*
         * if the sum xy + z will be exact, just compute it and cast the
         * result to float
         */
        exy = xy.i[2] & 0x7fff;
        ez = zz.i[2] & 0x7fff;

#endif /* ! codereview */
        if ((ez - exy <= 15 && exy - ez <= 39) || exy == 0x7fff || exy == 0 ||
            ez == 0x7fff || ez == 0)
```



```
                ez == 0x7fff || ez == 0) {
                goto cont;
        }

        /*
         * collapse the tail of the smaller summand into a "sticky bit"
         * so that the sum can be computed without error
         */
        if (ez > exy) {
                if (ez - exy < 31) {
                        u = xy.i[0];
                        s = 2 << (ez - exy);

#endif /* ! codereview */
                        if (u & (s - 1))
                                u |= s;

#endif /* ! codereview */
                        xy.i[0] = u & ~(s - 1);
                } else if (ez - exy < 62) {
                        u = xy.i[1];
                        s = 1 << (ez - exy - 31);

#endif /* ! codereview */
                        if ((u & (s - 1)) | xy.i[0])
                                u |= s;

#endif /* ! codereview */
                        xy.i[1] = u & ~(s - 1);
                        xy.i[0] = 0;
                } else {
                        /* collapse all of xy into a single bit */
                        xy.i[0] = 0;
                        xy.i[1] = 0x80000000;
                        xy.i[2] = (xy.i[2] & 0x8000) | (ez - 62);
                }
        } else {
                if (exy - ez < 62) {
                        u = zz.i[1];
                        s = 1 << (exy - ez - 31);

#endif /* ! codereview */
                        if ((u & (s - 1)) | zz.i[0])
                                u |= s;

#endif /* ! codereview */
                        zz.i[1] = u & ~(s - 1);
                        zz.i[0] = 0;
                } else {
                        /* collapse all of zz into a single bit */
                        zz.i[0] = 0;
                        zz.i[1] = 0x80000000;
                        zz.i[2] = (zz.i[2] & 0x8000) | (exy - 62);
                }
        }

cont:
        xy.e += zz.e;

        /* restore the rounding precision */
        __fenv_getcwsw(&cwsw);
        cwsw = (cwsw & 0xfcffffff) | (oldcwsw & 0x03000000);
        __fenv_setcwsw(&cwsw);

        return ((float)xy.e);
        return ((float) xy.e);
```

```
}

#if 0
/*
 * another fmaf for x86: assumes return value will be left in
 * long double (80-bit double extended) precision
 */
long double
__fmaf(float x, float y, float z)
{
__fmaf(float x, float y, float z) {
        /*
         * Note: This implementation assumes the rounding precision mode
         * is set to the default, rounding to 64 bit precision.  If this
         * routine must work in non-default rounding precision modes, do
         * the following instead:
         *
         *   long double t;
         *
         *   <set rp mode to round to 64 bit precision>
         *   t = x * y;
         *   <restore rp mode>
         *   return t + z;
         *
         * Note that the code to change rounding precision must not alter
         * the exception masks or flags, since the product x * y may raise
         * an invalid operation exception.
         */
        return ((long double)x * y + z);
        return ((long double) x * y + z);
}
#endif

#else
#error Unknown architecture
#endif
```

```
**********************************************************
   28412 Thu Jun  6 20:40:14 2019
new/usr/src/lib/libm/common/m9x/fmal.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak fmal = __fmal

#include "libm.h"
#include "fma.h"
#include "fenv_inlines.h"

#if defined(__sparc)

static const union {
        unsigned i[2];
        double d;
} C[] = {
______unchanged_portion_omitted_

#define half            C[0].d
#define two             C[1].d
#define twom16          C[2].d
#define twom24          C[3].d
#define two20           C[4].d
#define twom28          C[5].d
#define twom76          C[6].d
#define twom124         C[7].d
#define two36           C[8].d
#define twom32          C[9].d
#define huge            C[10].d
#define tiny            C[11].d
#define tiny2           C[12].d
#define zero            C[13].d
#define inf             C[14].d
#define snan            C[15].d
```



```
static const unsigned int fsr_rm = 0xc0000000u;

/*
 * fmal for SPARC: 128-bit quad precision, big-endian
 */
long double
__fmal(long double x, long double y, long double z)
{
__fmal(long double x, long double y, long double z) {
        union {
                unsigned int i[4];
                long double q;
        } xx, yy, zz;
        union {
                unsigned int i[2];
                double d;
        } u;

#endif /* ! codereview */
        double dx[5], dy[5], dxy[9], c, s;
        unsigned int xy0, xy1, xy2, xy3, xy4, xy5, xy6, xy7;
        unsigned int z0, z1, z2, z3, z4, z5, z6, z7;
        unsigned int rm, sticky;
        unsigned int fsr;
        int hx, hy, hz, ex, ey, ez, exy, sxy, sz, e, ibit;
        int cx, cy, cz;
        volatile double dummy;

        /* extract the high order words of the arguments */
        xx.q = x;
        yy.q = y;
        zz.q = z;
        hx = xx.i[0] & ~0x80000000;
        hy = yy.i[0] & ~0x80000000;
        hz = zz.i[0] & ~0x80000000;

        /*
         * distinguish zero, finite nonzero, infinite, and quiet nan
         * arguments; raise invalid and return for signaling nans
         */
        if (hx >= 0x7fff0000) {
                if ((hx & 0xffff) | xx.i[1] | xx.i[2] | xx.i[3]) {
                        if (!(hx & 0x8000)) {
                                /* signaling nan, raise invalid */
                                dummy = snan;
                                dummy += snan;
                                xx.i[0] |= 0x8000;
                                return (xx.q);
                        }

#endif /* ! codereview */
                        cx = 3;         /* quiet nan */
                } else {
                } else
                        cx = 2;         /* inf */
                }
#endif /* ! codereview */
        } else if (hx == 0) {
                cx = (xx.i[1] | xx.i[2] | xx.i[3]) ? 1 : 0;
                /* subnormal or zero */
        } else {
        } else
                cx = 1;                 /* finite nonzero */
        }
#endif /* ! codereview */
```

```
        if (hy >= 0x7fff0000) {
                if ((hy & 0xffff) | yy.i[1] | yy.i[2] | yy.i[3]) {
                        if (!(hy & 0x8000)) {
                                dummy = snan;
                                dummy += snan;
                                yy.i[0] |= 0x8000;
                                return (yy.q);
                        }

#endif /* ! codereview */
                        cy = 3;
                } else {
                } else
                        cy = 2;
                }
#endif /* ! codereview */
        } else if (hy == 0) {
                cy = (yy.i[1] | yy.i[2] | yy.i[3]) ? 1 : 0;
        } else {
        } else
                cy = 1;
        }
#endif /* ! codereview */

        if (hz >= 0x7fff0000) {
                if ((hz & 0xffff) | zz.i[1] | zz.i[2] | zz.i[3]) {
                        if (!(hz & 0x8000)) {
                                dummy = snan;
                                dummy += snan;
                                zz.i[0] |= 0x8000;
                                return (zz.q);
                        }

#endif /* ! codereview */
                        cz = 3;
                } else {
                } else
                        cz = 2;
                }
#endif /* ! codereview */
        } else if (hz == 0) {
                cz = (zz.i[1] | zz.i[2] | zz.i[3]) ? 1 : 0;
        } else {
        } else
                cz = 1;
        }
#endif /* ! codereview */

        /* get the fsr and clear current exceptions */
        __fenv_getfsr32(&fsr);
        fsr &= ~FSR_CEXC;

        /* handle all other zero, inf, and nan cases */
        if (cx != 1 || cy != 1 || cz != 1) {
                /* if x or y is a quiet nan, return it */
                if (cx == 3) {
                        __fenv_setfsr32(&fsr);
                        return (x);
                }

#endif /* ! codereview */
                if (cy == 3) {
                        __fenv_setfsr32(&fsr);
                        return (y);
                }
```



```
                /* if x*y is 0*inf, raise invalid and return the default nan */
                if ((cx == 0 && cy == 2) || (cx == 2 && cy == 0)) {
                        dummy = zero;
                        dummy *= inf;
                        zz.i[0] = 0x7fffffff;
                        zz.i[1] = zz.i[2] = zz.i[3] = 0xffffffff;
                        return (zz.q);
                }

                /* if z is a quiet nan, return it */
                if (cz == 3) {
                        __fenv_setfsr32(&fsr);
                        return (z);
                }

                /*
                 * now none of x, y, or z is nan; handle cases where x or y
                 * is inf
                 */
                if (cx == 2 || cy == 2) {
                        /*
                         * if z is also inf, either we have inf-inf or
                         * the result is the same as z depending on signs
                         */
                        if (cz == 2) {
                                if ((int)((xx.i[0] ^ yy.i[0]) ^ zz.i[0]) < 0) {
                                if ((int) ((xx.i[0] ^ yy.i[0]) ^ zz.i[0]) < 0) {
                                        dummy = inf;
                                        dummy -= inf;
                                        zz.i[0] = 0x7fffffff;
                                        zz.i[1] = zz.i[2] = zz.i[3] =
                                            0xffffffff;
                                        return (zz.q);
                                }
#endif /* ! codereview */
                                        __fenv_setfsr32(&fsr);
                                        return (z);
                        }

                        /* otherwise the result is inf with appropriate sign */
                        zz.i[0] = ((xx.i[0] ^ yy.i[0]) & 0x80000000) |
                            0x7fff0000;
                        zz.i[1] = zz.i[2] = zz.i[3] = 0;
                        __fenv_setfsr32(&fsr);
                        return (zz.q);
                }

                /* if z is inf, return it */
                if (cz == 2) {
                        __fenv_setfsr32(&fsr);
                        return (z);
                }

                /*
                 * now x, y, and z are all finite; handle cases where x or y
                 * is zero
                 */
                if (cx == 0 || cy == 0) {
                        /* either we have 0-0 or the result is the same as z */
                        if (cz == 0 && (int)((xx.i[0] ^ yy.i[0]) ^ zz.i[0]) <
                        if (cz == 0 && (int) ((xx.i[0] ^ yy.i[0]) ^ zz.i[0]) <
                            0) {
                                zz.i[0] = (fsr >> 30) == FSR_RM ? 0x80000000 :
                                    0;
```

```
                                __fenv_setfsr32(&fsr);
                                return (zz.q);
                        }

#endif /* ! codereview */
                        __fenv_setfsr32(&fsr);
                        return (z);
                }

                /* if we get here, x and y are nonzero finite, z must be zero */
                return (x * y);
        }

        /*
         * now x, y, and z are all finite and nonzero; set round-to-
         * negative-infinity mode
         */
        __fenv_setfsr32(&fsr_rm);

        /*
         * get the signs and exponents and normalize the significands
         * of x and y
         */
        sxy = (xx.i[0] ^ yy.i[0]) & 0x80000000;
        ex = hx >> 16;
        hx &= 0xffff;

#endif /* ! codereview */
        if (!ex) {
                if (hx | (xx.i[1] & 0xfffe0000)) {
                        ex = 1;
                } else if (xx.i[1] | (xx.i[2] & 0xfffe0000)) {
                        hx = xx.i[1];
                        xx.i[1] = xx.i[2];
                        xx.i[2] = xx.i[3];
                        xx.i[3] = 0;
                        ex = -31;
                } else if (xx.i[2] | (xx.i[3] & 0xfffe0000)) {
                        hx = xx.i[2];
                        xx.i[1] = xx.i[3];
                        xx.i[2] = xx.i[3] = 0;
                        ex = -63;
                } else {
                        hx = xx.i[3];
                        xx.i[1] = xx.i[2] = xx.i[3] = 0;
                        ex = -95;
                }

#endif /* ! codereview */
                while ((hx & 0x10000) == 0) {
                        hx = (hx << 1) | (xx.i[1] >> 31);
                        xx.i[1] = (xx.i[1] << 1) | (xx.i[2] >> 31);
                        xx.i[2] = (xx.i[2] << 1) | (xx.i[3] >> 31);
                        xx.i[3] <<= 1;
                        ex--;
                }
        } else {
        } else
                hx |= 0x10000;
        }

#endif /* ! codereview */
        ey = hy >> 16;
        hy &= 0xffff;

#endif /* ! codereview */
```



```
        if (!ey) {
                if (hy | (yy.i[1] & 0xfffe0000)) {
                        ey = 1;
                } else if (yy.i[1] | (yy.i[2] & 0xfffe0000)) {
                        hy = yy.i[1];
                        yy.i[1] = yy.i[2];
                        yy.i[2] = yy.i[3];
                        yy.i[3] = 0;
                        ey = -31;
                } else if (yy.i[2] | (yy.i[3] & 0xfffe0000)) {
                        hy = yy.i[2];
                        yy.i[1] = yy.i[3];
                        yy.i[2] = yy.i[3] = 0;
                        ey = -63;
                } else {
                        hy = yy.i[3];
                        yy.i[1] = yy.i[2] = yy.i[3] = 0;
                        ey = -95;
                }

#endif /* ! codereview */
                while ((hy & 0x10000) == 0) {
                        hy = (hy << 1) | (yy.i[1] >> 31);
                        yy.i[1] = (yy.i[1] << 1) | (yy.i[2] >> 31);
                        yy.i[2] = (yy.i[2] << 1) | (yy.i[3] >> 31);
                        yy.i[3] <<= 1;
                        ey--;
                }
        } else {
        } else
                hy |= 0x10000;
        }

#endif /* ! codereview */
        exy = ex + ey - 0x3fff;

        /* convert the significands of x and y to doubles */
        c = twom16;
        dx[0] = (double)((int)hx) * c;
        dy[0] = (double)((int)hy) * c;
        dx[0] = (double) ((int) hx) * c;
        dy[0] = (double) ((int) hy) * c;

        c *= twom24;
        dx[1] = (double)((int)(xx.i[1] >> 8)) * c;
        dy[1] = (double)((int)(yy.i[1] >> 8)) * c;
        dx[1] = (double) ((int) (xx.i[1] >> 8)) * c;
        dy[1] = (double) ((int) (yy.i[1] >> 8)) * c;

        c *= twom24;
        dx[2] = (double)((int)(((xx.i[1] << 16) | (xx.i[2] >> 16)) &
        dx[2] = (double) ((int) (((xx.i[1] << 16) | (xx.i[2] >> 16)) &
            0xffffff)) * c;
        dy[2] = (double)((int)(((yy.i[1] << 16) | (yy.i[2] >> 16)) &
        dy[2] = (double) ((int) (((yy.i[1] << 16) | (yy.i[2] >> 16)) &
            0xffffff)) * c;

        c *= twom24;
        dx[3] = (double)((int)(((xx.i[2] << 8) | (xx.i[3] >> 24)) & 0xffffff)) *
            c;
        dy[3] = (double)((int)(((yy.i[2] << 8) | (yy.i[3] >> 24)) & 0xffffff)) *
            c;
        dx[3] = (double) ((int) (((xx.i[2] << 8) | (xx.i[3] >> 24)) &
            0xffffff)) * c;
        dy[3] = (double) ((int) (((yy.i[2] << 8) | (yy.i[3] >> 24)) &
            0xffffff)) * c;
```

```
        c *= twom24;
        dx[4] = (double)((int)(xx.i[3] & 0xffffff)) * c;
        dy[4] = (double)((int)(yy.i[3] & 0xffffff)) * c;
        dx[4] = (double) ((int) (xx.i[3] & 0xffffff)) * c;
        dy[4] = (double) ((int) (yy.i[3] & 0xffffff)) * c;

        /* form the "digits" of the product */
        dxy[0] = dx[0] * dy[0];
        dxy[1] = dx[0] * dy[1] + dx[1] * dy[0];
        dxy[2] = dx[0] * dy[2] + dx[1] * dy[1] + dx[2] * dy[0];
        dxy[3] = dx[0] * dy[3] + dx[1] * dy[2] + dx[2] * dy[1] + dx[3] * dy[0];
        dxy[4] = dx[0] * dy[4] + dx[1] * dy[3] + dx[2] * dy[2] + dx[3] * dy[1] +
            dx[4] * dy[0];
        dxy[5] = dx[1] * dy[4] + dx[2] * dy[3] + dx[3] * dy[2] + dx[4] * dy[1];
        dxy[3] = dx[0] * dy[3] + dx[1] * dy[2] + dx[2] * dy[1] +
            dx[3] * dy[0];
        dxy[4] = dx[0] * dy[4] + dx[1] * dy[3] + dx[2] * dy[2] +
            dx[3] * dy[1] + dx[4] * dy[0];
        dxy[5] = dx[1] * dy[4] + dx[2] * dy[3] + dx[3] * dy[2] +
            dx[4] * dy[1];
        dxy[6] = dx[2] * dy[4] + dx[3] * dy[3] + dx[4] * dy[2];
        dxy[7] = dx[3] * dy[4] + dx[4] * dy[3];
        dxy[8] = dx[4] * dy[4];

        /* split odd-numbered terms and combine into even-numbered terms */
        c = (dxy[1] + two20) - two20;
        dxy[0] += c;
        dxy[1] -= c;
        c = (dxy[3] + twom28) - twom28;
        dxy[2] += c + dxy[1];
        dxy[3] -= c;
        c = (dxy[5] + twom76) - twom76;
        dxy[4] += c + dxy[3];
        dxy[5] -= c;
        c = (dxy[7] + twom124) - twom124;
        dxy[6] += c + dxy[5];
        dxy[8] += (dxy[7] - c);

        /* propagate carries, adjusting the exponent if need be */
        dxy[7] = dxy[6] + dxy[8];
        dxy[5] = dxy[4] + dxy[7];
        dxy[3] = dxy[2] + dxy[5];
        dxy[1] = dxy[0] + dxy[3];

#endif /* ! codereview */
        if (dxy[1] >= two) {
                dxy[0] *= half;
                dxy[1] *= half;
                dxy[2] *= half;
                dxy[3] *= half;
                dxy[4] *= half;
                dxy[5] *= half;
                dxy[6] *= half;
                dxy[7] *= half;
                dxy[8] *= half;
                exy++;
        }

        /* extract the significand of x*y */
        s = two36;
        u.d = c = dxy[1] + s;
        xy0 = u.i[1];
        c -= s;
        dxy[1] -= c;
        dxy[0] -= c;
```



```
        s *= twom32;
        u.d = c = dxy[1] + s;
        xy1 = u.i[1];
        c -= s;
        dxy[2] += (dxy[0] - c);
        dxy[3] = dxy[2] + dxy[5];

        s *= twom32;
        u.d = c = dxy[3] + s;
        xy2 = u.i[1];
        c -= s;
        dxy[4] += (dxy[2] - c);
        dxy[5] = dxy[4] + dxy[7];

        s *= twom32;
        u.d = c = dxy[5] + s;
        xy3 = u.i[1];
        c -= s;
        dxy[4] -= c;
        dxy[5] = dxy[4] + dxy[7];

        s *= twom32;
        u.d = c = dxy[5] + s;
        xy4 = u.i[1];
        c -= s;
        dxy[6] += (dxy[4] - c);
        dxy[7] = dxy[6] + dxy[8];

        s *= twom32;
        u.d = c = dxy[7] + s;
        xy5 = u.i[1];
        c -= s;
        dxy[8] += (dxy[6] - c);

        s *= twom32;
        u.d = c = dxy[8] + s;
        xy6 = u.i[1];
        c -= s;
        dxy[8] -= c;

        s *= twom32;
        u.d = c = dxy[8] + s;
        xy7 = u.i[1];

        /* extract the sign, exponent, and significand of z */
        sz = zz.i[0] & 0x80000000;
        ez = hz >> 16;
        z0 = hz & 0xffff;

#endif /* ! codereview */
        if (!ez) {
                if (z0 | (zz.i[1] & 0xfffe0000)) {
                        z1 = zz.i[1];
                        z2 = zz.i[2];
                        z3 = zz.i[3];
                        ez = 1;
                } else if (zz.i[1] | (zz.i[2] & 0xfffe0000)) {
                        z0 = zz.i[1];
                        z1 = zz.i[2];
                        z2 = zz.i[3];
                        z3 = 0;
                        ez = -31;
                } else if (zz.i[2] | (zz.i[3] & 0xfffe0000)) {
                        z0 = zz.i[2];
                        z1 = zz.i[3];
```

```
                        z2 = z3 = 0;
                        ez = -63;
                } else {
                        z0 = zz.i[3];
                        z1 = z2 = z3 = 0;
                        ez = -95;
                }

#endif /* ! codereview */
                while ((z0 & 0x10000) == 0) {
                        z0 = (z0 << 1) | (z1 >> 31);
                        z1 = (z1 << 1) | (z2 >> 31);
                        z2 = (z2 << 1) | (z3 >> 31);
                        z3 <<= 1;
                        ez--;
                }
        } else {
                z0 |= 0x10000;
                z1 = zz.i[1];
                z2 = zz.i[2];
                z3 = zz.i[3];
        }

#endif /* ! codereview */
        z4 = z5 = z6 = z7 = 0;

        /*
         * now x*y is represented by sxy, exy, and xy[0-7], and z is
         * represented likewise; swap if need be so |xy| <= |z|
         */
        if (exy > ez || (exy == ez && (xy0 > z0 || (xy0 == z0 && (xy1 > z1 ||
            (xy1 == z1 && (xy2 > z2 || (xy2 == z2 && (xy3 > z3 || (xy3 == z3 &&
            (xy4 | xy5 | xy6 | xy7) != 0))))))))) {
                e = sxy;
                sxy = sz;
                sz = e;
                e = exy;
                exy = ez;
                ez = e;
                e = xy0;
                xy0 = z0;
                z0 = e;
                e = xy1;
                xy1 = z1;
                z1 = e;
                e = xy2;
                xy2 = z2;
                z2 = e;
                e = xy3;
                xy3 = z3;
                z3 = e;
                z4 = xy4;
                xy4 = 0;
                z5 = xy5;
                xy5 = 0;
                z6 = xy6;
                xy6 = 0;
                z7 = xy7;
                xy7 = 0;
                (xy1 == z1 && (xy2 > z2 || (xy2 == z2 && (xy3 > z3 ||
                (xy3 == z3 && (xy4 | xy5 | xy6 | xy7) != 0))))))))) {
                e = sxy; sxy = sz; sz = e;
                e = exy; exy = ez; ez = e;
                e = xy0; xy0 = z0; z0 = e;
                e = xy1; xy1 = z1; z1 = e;
                e = xy2; xy2 = z2; z2 = e;
```



```
                e = xy3; xy3 = z3; z3 = e;
                z4 = xy4; xy4 = 0;
                z5 = xy5; xy5 = 0;
                z6 = xy6; xy6 = 0;
                z7 = xy7; xy7 = 0;
        }

        /* shift the significand of xy keeping a sticky bit */
        e = ez - exy;

#endif /* ! codereview */
        if (e > 236) {
                xy0 = xy1 = xy2 = xy3 = xy4 = xy5 = xy6 = 0;
                xy7 = 1;
        } else if (e >= 224) {
                sticky = xy7 | xy6 | xy5 | xy4 | xy3 | xy2 | xy1 | ((xy0 <<
                    1) << (255 - e));
                sticky = xy7 | xy6 | xy5 | xy4 | xy3 | xy2 | xy1 |
                        ((xy0 << 1) << (255 - e));
                xy7 = xy0 >> (e - 224);

#endif /* ! codereview */
                if (sticky)
                        xy7 |= 1;

#endif /* ! codereview */
                xy0 = xy1 = xy2 = xy3 = xy4 = xy5 = xy6 = 0;
        } else if (e >= 192) {
                sticky = xy7 | xy6 | xy5 | xy4 | xy3 | xy2 | ((xy1 << 1) <<
                    (223 - e));
                sticky = xy7 | xy6 | xy5 | xy4 | xy3 | xy2 |
                        ((xy1 << 1) << (223 - e));
                xy7 = (xy1 >> (e - 192)) | ((xy0 << 1) << (223 - e));

#endif /* ! codereview */
                if (sticky)
                        xy7 |= 1;

#endif /* ! codereview */
                xy6 = xy0 >> (e - 192);
                xy0 = xy1 = xy2 = xy3 = xy4 = xy5 = 0;
        } else if (e >= 160) {
                sticky = xy7 | xy6 | xy5 | xy4 | xy3 | ((xy2 << 1) << (191 -
                    e));
                sticky = xy7 | xy6 | xy5 | xy4 | xy3 |
                        ((xy2 << 1) << (191 - e));
                xy7 = (xy2 >> (e - 160)) | ((xy1 << 1) << (191 - e));

#endif /* ! codereview */
                if (sticky)
                        xy7 |= 1;

#endif /* ! codereview */
                xy6 = (xy1 >> (e - 160)) | ((xy0 << 1) << (191 - e));
                xy5 = xy0 >> (e - 160);
                xy0 = xy1 = xy2 = xy3 = xy4 = 0;
        } else if (e >= 128) {
                sticky = xy7 | xy6 | xy5 | xy4 | ((xy3 << 1) << (159 - e));
                xy7 = (xy3 >> (e - 128)) | ((xy2 << 1) << (159 - e));

#endif /* ! codereview */
                if (sticky)
                        xy7 |= 1;

#endif /* ! codereview */
                xy6 = (xy2 >> (e - 128)) | ((xy1 << 1) << (159 - e));
```

```
                xy5 = (xy1 >> (e - 128)) | ((xy0 << 1) << (159 - e));
                xy4 = xy0 >> (e - 128);
                xy0 = xy1 = xy2 = xy3 = 0;
        } else if (e >= 96) {
                sticky = xy7 | xy6 | xy5 | ((xy4 << 1) << (127 - e));
                xy7 = (xy4 >> (e - 96)) | ((xy3 << 1) << (127 - e));

#endif /* ! codereview */
                if (sticky)
                        xy7 |= 1;

#endif /* ! codereview */
                xy6 = (xy3 >> (e - 96)) | ((xy2 << 1) << (127 - e));
                xy5 = (xy2 >> (e - 96)) | ((xy1 << 1) << (127 - e));
                xy4 = (xy1 >> (e - 96)) | ((xy0 << 1) << (127 - e));
                xy3 = xy0 >> (e - 96);
                xy0 = xy1 = xy2 = 0;
        } else if (e >= 64) {
                sticky = xy7 | xy6 | ((xy5 << 1) << (95 - e));
                xy7 = (xy5 >> (e - 64)) | ((xy4 << 1) << (95 - e));

#endif /* ! codereview */
                if (sticky)
                        xy7 |= 1;

#endif /* ! codereview */
                xy6 = (xy4 >> (e - 64)) | ((xy3 << 1) << (95 - e));
                xy5 = (xy3 >> (e - 64)) | ((xy2 << 1) << (95 - e));
                xy4 = (xy2 >> (e - 64)) | ((xy1 << 1) << (95 - e));
                xy3 = (xy1 >> (e - 64)) | ((xy0 << 1) << (95 - e));
                xy2 = xy0 >> (e - 64);
                xy0 = xy1 = 0;
        } else if (e >= 32) {
                sticky = xy7 | ((xy6 << 1) << (63 - e));
                xy7 = (xy6 >> (e - 32)) | ((xy5 << 1) << (63 - e));

#endif /* ! codereview */
                if (sticky)
                        xy7 |= 1;

#endif /* ! codereview */
                xy6 = (xy5 >> (e - 32)) | ((xy4 << 1) << (63 - e));
                xy5 = (xy4 >> (e - 32)) | ((xy3 << 1) << (63 - e));
                xy4 = (xy3 >> (e - 32)) | ((xy2 << 1) << (63 - e));
                xy3 = (xy2 >> (e - 32)) | ((xy1 << 1) << (63 - e));
                xy2 = (xy1 >> (e - 32)) | ((xy0 << 1) << (63 - e));
                xy1 = xy0 >> (e - 32);
                xy0 = 0;
        } else if (e) {
                sticky = (xy7 << 1) << (31 - e);
                xy7 = (xy7 >> e) | ((xy6 << 1) << (31 - e));

#endif /* ! codereview */
                if (sticky)
                        xy7 |= 1;

#endif /* ! codereview */
                xy6 = (xy6 >> e) | ((xy5 << 1) << (31 - e));
                xy5 = (xy5 >> e) | ((xy4 << 1) << (31 - e));
                xy4 = (xy4 >> e) | ((xy3 << 1) << (31 - e));
                xy3 = (xy3 >> e) | ((xy2 << 1) << (31 - e));
                xy2 = (xy2 >> e) | ((xy1 << 1) << (31 - e));
                xy1 = (xy1 >> e) | ((xy0 << 1) << (31 - e));
                xy0 >>= e;
        }
```



```
        /* if this is a magnitude subtract, negate the significand of xy */
        if (sxy ^ sz) {
                xy0 = ~xy0;
                xy1 = ~xy1;
                xy2 = ~xy2;
                xy3 = ~xy3;
                xy4 = ~xy4;
                xy5 = ~xy5;
                xy6 = ~xy6;
                xy7 = -xy7;

#endif /* ! codereview */
                if (xy7 == 0)
                        if (++xy6 == 0)
                                if (++xy5 == 0)
                                        if (++xy4 == 0)
                                                if (++xy3 == 0)
                                                        if (++xy2 == 0)
                                                                if (++xy1 == 0)
                                                                        xy0++;
        }

        /* add, propagating carries */
        z7 += xy7;
        e = (z7 < xy7);
        z6 += xy6;

#endif /* ! codereview */
        if (e) {
                z6++;
                e = (z6 <= xy6);
        } else {
        } else
                e = (z6 < xy6);
        }

#endif /* ! codereview */
        z5 += xy5;

#endif /* ! codereview */
        if (e) {
                z5++;
                e = (z5 <= xy5);
        } else {
        } else
                e = (z5 < xy5);
        }

#endif /* ! codereview */
        z4 += xy4;

#endif /* ! codereview */
        if (e) {
                z4++;
                e = (z4 <= xy4);
        } else {
        } else
                e = (z4 < xy4);
        }

#endif /* ! codereview */
        z3 += xy3;

#endif /* ! codereview */
        if (e) {
                z3++;
```

```
753                 e = (z3 <= xy3);
754         } else {
198         } else
755                 e = (z3 < xy3);
756         }

758 #endif /* ! codereview */
759         z2 += xy2;

761 #endif /* ! codereview */
762         if (e) {
763                 z2++;
764                 e = (z2 <= xy2);
765         } else {
200         } else
766                 e = (z2 < xy2);
767         }

769 #endif /* ! codereview */
770         z1 += xy1;

772 #endif /* ! codereview */
773         if (e) {
774                 z1++;
775                 e = (z1 <= xy1);
776         } else {
202         } else
777                 e = (z1 < xy1);
778         }

780 #endif /* ! codereview */
781         z0 += xy0;

783 #endif /* ! codereview */
784         if (e)
785                 z0++;

787         /* postnormalize and collect rounding information into z4 */
788         if (ez < 1) {
789                 /* result is tiny; shift right until exponent is within range */
790                 e = 1 - ez;

792 #endif /* ! codereview */
793                 if (e > 116) {
794                         z4 = 1;         /* result can't be exactly zero */
795                         z0 = z1 = z2 = z3 = 0;
796                 } else if (e >= 96) {
797                         sticky = z7 | z6 | z5 | z4 | z3 | z2 | ((z1 << 1) <<
798                             (127 - e));
204                         sticky = z7 | z6 | z5 | z4 | z3 | z2 |
205                                 ((z1 << 1) << (127 - e));
799                         z4 = (z1 >> (e - 96)) | ((z0 << 1) << (127 - e));

801 #endif /* ! codereview */
802                         if (sticky)
803                                 z4 |= 1;

805 #endif /* ! codereview */
806                         z3 = z0 >> (e - 96);
807                         z0 = z1 = z2 = 0;
808                 } else if (e >= 64) {
809                         sticky = z7 | z6 | z5 | z4 | z3 | ((z2 << 1) << (95 -
810                             e));
207                         sticky = z7 | z6 | z5 | z4 | z3 |
208                                 ((z2 << 1) << (95 - e));
811                         z4 = (z2 >> (e - 64)) | ((z1 << 1) << (95 - e));
```



```
813 #endif /* ! codereview */
814                         if (sticky)
815                                 z4 |= 1;

817 #endif /* ! codereview */
818                         z3 = (z1 >> (e - 64)) | ((z0 << 1) << (95 - e));
819                         z2 = z0 >> (e - 64);
820                         z0 = z1 = 0;
821                 } else if (e >= 32) {
822                         sticky = z7 | z6 | z5 | z4 | ((z3 << 1) << (63 - e));
823                         z4 = (z3 >> (e - 32)) | ((z2 << 1) << (63 - e));

825 #endif /* ! codereview */
826                         if (sticky)
827                                 z4 |= 1;

829 #endif /* ! codereview */
830                         z3 = (z2 >> (e - 32)) | ((z1 << 1) << (63 - e));
831                         z2 = (z1 >> (e - 32)) | ((z0 << 1) << (63 - e));
832                         z1 = z0 >> (e - 32);
833                         z0 = 0;
834                 } else {
835                         sticky = z7 | z6 | z5 | (z4 << 1) << (31 - e);
836                         z4 = (z4 >> e) | ((z3 << 1) << (31 - e));

838 #endif /* ! codereview */
839                         if (sticky)
840                                 z4 |= 1;

842 #endif /* ! codereview */
843                         z3 = (z3 >> e) | ((z2 << 1) << (31 - e));
844                         z2 = (z2 >> e) | ((z1 << 1) << (31 - e));
845                         z1 = (z1 >> e) | ((z0 << 1) << (31 - e));
846                         z0 >>= e;
847                 }

849 #endif /* ! codereview */
850                 ez = 1;
851         } else if (z0 >= 0x20000) {
852                 /* carry out; shift right by one */
853                 sticky = (z4 & 1) | z5 | z6 | z7;
854                 z4 = (z4 >> 1) | (z3 << 31);

856 #endif /* ! codereview */
857                 if (sticky)
858                         z4 |= 1;

860 #endif /* ! codereview */
861                 z3 = (z3 >> 1) | (z2 << 31);
862                 z2 = (z2 >> 1) | (z1 << 31);
863                 z1 = (z1 >> 1) | (z0 << 31);
864                 z0 >>= 1;
865                 ez++;
866         } else {
867                 if (z0 < 0x10000 && (z0 | z1 | z2 | z3 | z4 | z5 | z6 | z7) !=
868                     0) {
210                 if (z0 < 0x10000 && (z0 | z1 | z2 | z3 | z4 | z5 | z6 | z7)
211                         != 0) {
869                         /*
870                          * borrow/cancellation; shift left as much as
871                          * exponent allows
872                          */
873                         while (!(z0 | (z1 & 0xfffe0000)) && ez >= 33) {
874                                 z0 = z1;
875                                 z1 = z2;
```

```
                                z2 = z3;
                                z3 = z4;
                                z4 = z5;
                                z5 = z6;
                                z6 = z7;
                                z7 = 0;
                                ez -= 32;
                        }

#endif /* ! codereview */
                        while (z0 < 0x10000 && ez > 1) {
                                z0 = (z0 << 1) | (z1 >> 31);
                                z1 = (z1 << 1) | (z2 >> 31);
                                z2 = (z2 << 1) | (z3 >> 31);
                                z3 = (z3 << 1) | (z4 >> 31);
                                z4 = (z4 << 1) | (z5 >> 31);
                                z5 = (z5 << 1) | (z6 >> 31);
                                z6 = (z6 << 1) | (z7 >> 31);
                                z7 <<= 1;
                                ez--;
                        }
                }

#endif /* ! codereview */
                if (z5 | z6 | z7)
                        z4 |= 1;
        }

        /* get the rounding mode */
        rm = fsr >> 30;

        /* strip off the integer bit, if there is one */
        ibit = z0 & 0x10000;

        if (ibit) {
        if (ibit)
                z0 -= 0x10000;
        } else {
        else {
                ez = 0;

#endif /* ! codereview */
                if (!(z0 | z1 | z2 | z3 | z4)) {        /* exact zero */
                        zz.i[0] = rm == FSR_RM ? 0x80000000 : 0;
                        zz.i[1] = zz.i[2] = zz.i[3] = 0;
                        __fenv_setfsr32(&fsr);
                        return (zz.q);
                }
        }

        /*
         * flip the sense of directed roundings if the result is negative;
         * the logic below applies to a positive result
         */
        if (sz)
                rm ^= rm >> 1;

        /* round and raise exceptions */
        if (z4) {
                fsr |= FSR_NXC;

                /* decide whether to round the fraction up */
                if (rm == FSR_RP || (rm == FSR_RN && (z4 > 0x80000000u || (z4 ==
                    0x80000000u && (z3 & 1))))) {
                if (rm == FSR_RP || (rm == FSR_RN && (z4 > 0x80000000u ||
                        (z4 == 0x80000000u && (z3 & 1))))) {
```



```
                        /* round up and renormalize if necessary */
                        if (++z3 == 0)
                                if (++z2 == 0)
                                        if (++z1 == 0)
                                                if (++z0 == 0x10000) {
                                                        z0 = 0;
                                                        ez++;
                                                }
                }
        }

        /* check for under/overflow */
        if (ez >= 0x7fff) {
                if (rm == FSR_RN || rm == FSR_RP) {
                        zz.i[0] = sz | 0x7fff0000;
                        zz.i[1] = zz.i[2] = zz.i[3] = 0;
                } else {
                        zz.i[0] = sz | 0x7ffeffff;
                        zz.i[1] = zz.i[2] = zz.i[3] = 0xffffffff;
                }

#endif /* ! codereview */
                fsr |= FSR_OFC | FSR_NXC;
        } else {
                zz.i[0] = sz | (ez << 16) | z0;
                zz.i[1] = z1;
                zz.i[2] = z2;
                zz.i[3] = z3;

                /*
                 * !ibit => exact result was tiny before rounding,
                 * z4 nonzero => result delivered is inexact
                 */
                if (!ibit) {
                        if (z4)
                                fsr |= FSR_UFC | FSR_NXC;
                        else if (fsr & FSR_UFM)
                                fsr |= FSR_UFC;
                }
        }

        /* restore the fsr and emulate exceptions as needed */
        if ((fsr & FSR_CEXC) & (fsr >> 23)) {
                __fenv_setfsr32(&fsr);

#endif /* ! codereview */
                if (fsr & FSR_OFC) {
                        dummy = huge;
                        dummy *= huge;
                } else if (fsr & FSR_UFC) {
                        dummy = tiny;

#endif /* ! codereview */
                        if (fsr & FSR_NXC)
                                dummy *= tiny;
                        else
                                dummy -= tiny2;
                } else {
                        dummy = huge;
                        dummy += tiny;
                }
        } else {
                fsr |= (fsr & 0x1f) << 5;
                __fenv_setfsr32(&fsr);
        }
```

```
#endif /* ! codereview */
        return (zz.q);
}

#elif defined(__x86)

static const union {
        unsigned i[2];
        double d;
} C[] = {
______unchanged_portion_omitted_

#define half            C[0].d
#define two             C[1].d
#define twom32          C[2].d
#define twom64          C[3].d
#define two32           C[4].d
#define two63           C[5].d
#define huge            C[6].d
#define tiny            C[7].d
#define tiny2           C[8].d

#if defined(__amd64)
#define NI              4
#else
#define NI              3
#endif

/*
 * fmal for x86: 80-bit extended double precision, little-endian
 */
long double
__fmal(long double x, long double y, long double z)
{
__fmal(long double x, long double y, long double z) {
        union {
                unsigned i[NI];
                long double e;
        } xx, yy, zz;

#endif /* ! codereview */
        long double xhi, yhi, xlo, ylo, t;
        unsigned xy0, xy1, xy2, xy3, xy4, z0, z1, z2, z3, z4;
        unsigned oldcwsw, cwsw, rm, sticky, carry;
        int ex, ey, ez, exy, sxy, sz, e, tinyafter;
        volatile double dummy;

        /* extract the exponents of the arguments */
        xx.e = x;
        yy.e = y;
        zz.e = z;
        ex = xx.i[2] & 0x7fff;
        ey = yy.i[2] & 0x7fff;
        ez = zz.i[2] & 0x7fff;

        /* dispense with inf, nan, and zero cases */
        if (ex == 0x7fff || ey == 0x7fff || (ex | xx.i[1] | xx.i[0]) == 0 ||
            (ey | yy.i[1] | yy.i[0]) == 0)      /* x or y is inf, nan, or 0 */
                return (x * y + z);

        if (ez == 0x7fff)       /* z is inf or nan */
                return (x + z); /* avoid spurious under/overflow in x * y */

        if ((ez | zz.i[1] | zz.i[0]) == 0)      /* z is zero */
                /*
                 * x * y isn't zero but could underflow to zero,
```



```
                 * so don't add z, lest we perturb the sign
                 */
                return (x * y);

        /*
         * now x, y, and z are all finite and nonzero; extract signs and
         * normalize the significands (this will raise the denormal operand
         * exception if need be)
         */
        sxy = (xx.i[2] ^ yy.i[2]) & 0x8000;
        sz = zz.i[2] & 0x8000;

#endif /* ! codereview */
        if (!ex) {
                xx.e = x * two63;
                ex = (xx.i[2] & 0x7fff) - 63;
        }

#endif /* ! codereview */
        if (!ey) {
                yy.e = y * two63;
                ey = (yy.i[2] & 0x7fff) - 63;
        }

#endif /* ! codereview */
        if (!ez) {
                zz.e = z * two63;
                ez = (zz.i[2] & 0x7fff) - 63;
        }

        /*
         * save the control and status words, mask all exceptions, and
         * set rounding to 64-bit precision and toward-zero
         */
        __fenv_getcwsw(&oldcwsw);
        cwsw = (oldcwsw & 0xf0c0ffff) | 0x0f3f0000;
        __fenv_setcwsw(&cwsw);

        /* multiply x*y to 128 bits */
        exy = ex + ey - 0x3fff;
        xx.i[2] = 0x3fff;
        yy.i[2] = 0x3fff;
        x = xx.e;
        y = yy.e;
        xhi = ((x + twom32) + two32) - two32;
        yhi = ((y + twom32) + two32) - two32;
        xlo = x - xhi;
        ylo = y - yhi;
        x *= y;
        y = ((xhi * yhi - x) + xhi * ylo + xlo * yhi) + xlo * ylo;

#endif /* ! codereview */
        if (x >= two) {
                x *= half;
                y *= half;
                exy++;
        }

        /* extract the significands */
        xx.e = x;
        xy0 = xx.i[1];
        xy1 = xx.i[0];
        yy.e = t = y + twom32;
        xy2 = yy.i[0];
        yy.e = (y - (t - twom32)) + twom64;
        xy3 = yy.i[0];
```

```
        xy4 = 0;
        z0 = zz.i[1];
        z1 = zz.i[0];
        z2 = z3 = z4 = 0;

        /*
         * now x*y is represented by sxy, exy, and xy[0-4], and z is
         * represented likewise; swap if need be so |xy| <= |z|
         */
        if (exy > ez || (exy == ez && (xy0 > z0 || (xy0 == z0 && (xy1 > z1 ||
            (xy1 == z1 && (xy2 | xy3) != 0)))))) {
                e = sxy;
                sxy = sz;
                sz = e;
                e = exy;
                exy = ez;
                ez = e;
                e = xy0;
                xy0 = z0;
                z0 = e;
                e = xy1;
                xy1 = z1;
                z1 = e;
                z2 = xy2;
                xy2 = 0;
                z3 = xy3;
                xy3 = 0;
        if (exy > ez || (exy == ez && (xy0 > z0 || (xy0 == z0 &&
                (xy1 > z1 || (xy1 == z1 && (xy2 | xy3) != 0)))))) {
                e = sxy; sxy = sz; sz = e;
                e = exy; exy = ez; ez = e;
                e = xy0; xy0 = z0; z0 = e;
                e = xy1; xy1 = z1; z1 = e;
                z2 = xy2; xy2 = 0;
                z3 = xy3; xy3 = 0;
        }

        /* shift the significand of xy keeping a sticky bit */
        e = ez - exy;

#endif /* ! codereview */
        if (e > 130) {
                xy0 = xy1 = xy2 = xy3 = 0;
                xy4 = 1;
        } else if (e >= 128) {
                sticky = xy3 | xy2 | xy1 | ((xy0 << 1) << (159 - e));
                xy4 = xy0 >> (e - 128);

#endif /* ! codereview */
                if (sticky)
                        xy4 |= 1;

#endif /* ! codereview */
                xy0 = xy1 = xy2 = xy3 = 0;
        } else if (e >= 96) {
                sticky = xy3 | xy2 | ((xy1 << 1) << (127 - e));
                xy4 = (xy1 >> (e - 96)) | ((xy0 << 1) << (127 - e));

#endif /* ! codereview */
                if (sticky)
                        xy4 |= 1;

#endif /* ! codereview */
                xy3 = xy0 >> (e - 96);
                xy0 = xy1 = xy2 = 0;
        } else if (e >= 64) {
```



```
                sticky = xy3 | ((xy2 << 1) << (95 - e));
                xy4 = (xy2 >> (e - 64)) | ((xy1 << 1) << (95 - e));

#endif /* ! codereview */
                if (sticky)
                        xy4 |= 1;

#endif /* ! codereview */
                xy3 = (xy1 >> (e - 64)) | ((xy0 << 1) << (95 - e));
                xy2 = xy0 >> (e - 64);
                xy0 = xy1 = 0;
        } else if (e >= 32) {
                sticky = (xy3 << 1) << (63 - e);
                xy4 = (xy3 >> (e - 32)) | ((xy2 << 1) << (63 - e));

#endif /* ! codereview */
                if (sticky)
                        xy4 |= 1;

#endif /* ! codereview */
                xy3 = (xy2 >> (e - 32)) | ((xy1 << 1) << (63 - e));
                xy2 = (xy1 >> (e - 32)) | ((xy0 << 1) << (63 - e));
                xy1 = xy0 >> (e - 32);
                xy0 = 0;
        } else if (e) {
                xy4 = (xy3 << 1) << (31 - e);
                xy3 = (xy3 >> e) | ((xy2 << 1) << (31 - e));
                xy2 = (xy2 >> e) | ((xy1 << 1) << (31 - e));
                xy1 = (xy1 >> e) | ((xy0 << 1) << (31 - e));
                xy0 >>= e;
        }

        /* if this is a magnitude subtract, negate the significand of xy */
        if (sxy ^ sz) {
                xy0 = ~xy0;
                xy1 = ~xy1;
                xy2 = ~xy2;
                xy3 = ~xy3;
                xy4 = -xy4;

#endif /* ! codereview */
                if (xy4 == 0)
                        if (++xy3 == 0)
                                if (++xy2 == 0)
                                        if (++xy1 == 0)
                                                xy0++;
        }

        /* add, propagating carries */
        z4 += xy4;
        carry = (z4 < xy4);
        z3 += xy3;

#endif /* ! codereview */
        if (carry) {
                z3++;
                carry = (z3 <= xy3);
        } else {
        } else
                carry = (z3 < xy3);
        }

#endif /* ! codereview */
        z2 += xy2;

#endif /* ! codereview */
```

```
        if (carry) {
                z2++;
                carry = (z2 <= xy2);
        } else {
        } else
                carry = (z2 < xy2);
        }

#endif /* ! codereview */
        z1 += xy1;

#endif /* ! codereview */
        if (carry) {
                z1++;
                carry = (z1 <= xy1);
        } else {
        } else
                carry = (z1 < xy1);
        }

#endif /* ! codereview */
        z0 += xy0;

#endif /* ! codereview */
        if (carry) {
                z0++;
                carry = (z0 <= xy0);
        } else {
        } else
                carry = (z0 < xy0);
        }
#endif /* ! codereview */

        /* for a magnitude subtract, ignore the last carry out */
        if (sxy ^ sz)
                carry = 0;

        /* postnormalize and collect rounding information into z2 */
        if (ez < 1) {
                /* result is tiny; shift right until exponent is within range */
                e = 1 - ez;

#endif /* ! codereview */
                if (e > 67) {
                        z2 = 1;         /* result can't be exactly zero */
                        z0 = z1 = 0;
                } else if (e >= 64) {
                        sticky = z4 | z3 | z2 | z1 | ((z0 << 1) << (95 - e));
                        z2 = (z0 >> (e - 64)) | ((carry << 1) << (95 - e));

#endif /* ! codereview */
                        if (sticky)
                                z2 |= 1;

#endif /* ! codereview */
                        z1 = carry >> (e - 64);
                        z0 = 0;
                } else if (e >= 32) {
                        sticky = z4 | z3 | z2 | ((z1 << 1) << (63 - e));
                        z2 = (z1 >> (e - 32)) | ((z0 << 1) << (63 - e));

#endif /* ! codereview */
                        if (sticky)
                                z2 |= 1;

#endif /* ! codereview */
```



```
                        z1 = (z0 >> (e - 32)) | ((carry << 1) << (63 - e));
                        z0 = carry >> (e - 32);
                } else {
                        sticky = z4 | z3 | (z2 << 1) << (31 - e);
                        z2 = (z2 >> e) | ((z1 << 1) << (31 - e));

#endif /* ! codereview */
                        if (sticky)
                                z2 |= 1;

#endif /* ! codereview */
                        z1 = (z1 >> e) | ((z0 << 1) << (31 - e));
                        z0 = (z0 >> e) | ((carry << 1) << (31 - e));
                }

#endif /* ! codereview */
                ez = 1;
        } else if (carry) {
                /* carry out; shift right by one */
                sticky = (z2 & 1) | z3 | z4;
                z2 = (z2 >> 1) | (z1 << 31);

#endif /* ! codereview */
                if (sticky)
                        z2 |= 1;

#endif /* ! codereview */
                z1 = (z1 >> 1) | (z0 << 31);
                z0 = (z0 >> 1) | 0x80000000;
                ez++;
        } else {
                if (z0 < 0x80000000u && (z0 | z1 | z2 | z3 | z4) != 0) {
                        /*
                         * borrow/cancellation; shift left as much as
                         * exponent allows
                         */
                        while (!z0 && ez >= 33) {
                                z0 = z1;
                                z1 = z2;
                                z2 = z3;
                                z3 = z4;
                                z4 = 0;
                                ez -= 32;
                        }

#endif /* ! codereview */
                        while (z0 < 0x80000000u && ez > 1) {
                                z0 = (z0 << 1) | (z1 >> 31);
                                z1 = (z1 << 1) | (z2 >> 31);
                                z2 = (z2 << 1) | (z3 >> 31);
                                z3 = (z3 << 1) | (z4 >> 31);
                                z4 <<= 1;
                                ez--;
                        }
                }

#endif /* ! codereview */
                if (z3 | z4)
                        z2 |= 1;
        }

        /* get the rounding mode */
        rm = oldcwsw & 0x0c000000;

        /* adjust exponent if result is subnormal */
        tinyafter = 0;
```

```
#endif /* ! codereview */
        if (!(z0 & 0x80000000)) {
                ez = 0;
                tinyafter = 1;

#endif /* ! codereview */
                if (!(z0 | z1 | z2)) {  /* exact zero */
                        zz.i[2] = rm == FCW_RM ? 0x8000 : 0;
                        zz.i[1] = zz.i[0] = 0;
                        __fenv_setcwsw(&oldcwsw);
                        return (zz.e);
                }
        }

        /*
         * flip the sense of directed roundings if the result is negative;
         * the logic below applies to a positive result
         */
        if (sz && (rm == FCW_RM || rm == FCW_RP))
                rm = (FCW_RM + FCW_RP) - rm;

        /* round */
        if (z2) {
                if (rm == FCW_RP || (rm == FCW_RN && (z2 > 0x80000000u || (z2 ==
                    0x80000000u && (z1 & 1))))) {
                if (rm == FCW_RP || (rm == FCW_RN && (z2 > 0x80000000u ||
                        (z2 == 0x80000000u && (z1 & 1))))) {
                        /* round up and renormalize if necessary */
                        if (++z1 == 0) {
                                if (++z0 == 0) {
                                        z0 = 0x80000000;
                                        ez++;
                                } else if (z0 == 0x80000000) {
                                        /* rounded up to smallest normal */
                                        ez = 1;
```

```
#endif /* ! codereview */
                                        if ((rm == FCW_RP && z2 >
                                            0x80000000u) || (rm == FCW_RN &&
                                            z2 >= 0xc0000000u))
                                                /*
                                                 * would have rounded up to
                                                 * smallest normal even with
                                                 * unbounded range
                                                 */
                                                tinyafter = 0;
                                }
                        }
                }
        }

        /* restore the control and status words, check for over/underflow */
        __fenv_setcwsw(&oldcwsw);

#endif /* ! codereview */
        if (ez >= 0x7fff) {
                if (rm == FCW_RN || rm == FCW_RP) {
                        zz.i[2] = sz | 0x7fff;
                        zz.i[1] = 0x80000000;
                        zz.i[0] = 0;
                } else {
                        zz.i[2] = sz | 0x7ffe;
                        zz.i[1] = 0xffffffff;
                        zz.i[0] = 0xffffffff;
                }
```



```
#endif /* ! codereview */
                        dummy = huge;
                        dummy *= huge;
                } else {
                        zz.i[2] = sz | ez;
                        zz.i[1] = z0;
                        zz.i[0] = z1;

                        /*
                         * tinyafter => result rounded w/ unbounded range would be tiny,
                         * z2 nonzero => result delivered is inexact
                         */
                        if (tinyafter) {
                                dummy = tiny;

#endif /* ! codereview */
                                if (z2)
                                        dummy *= tiny;
                                else
                                        dummy -= tiny2;
                        } else if (z2) {
                                dummy = huge;
                                dummy += tiny;
                        }
                }

                return (zz.e);
}

#else
#error Unknown architecture
#endif
```

```
**********************************************************
    1986 Thu Jun  6 20:40:14 2019
new/usr/src/lib/libm/common/m9x/fmax.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak fmax = __fmax

/*
 * fmax(x,y) returns the larger of x and y.  If just one of the
 * arguments is NaN, fmax returns the other argument.  If both
 * arguments are NaN, fmax returns NaN.
 *
 * See fmaxf.c for a discussion of implementation trade-offs.
 */

#include "libm.h"                       /* for isgreaterequal macro */
#include <fenv.h>

double
__fmax(double x, double y)
{
__fmax(double x, double y) {
        union {
                unsigned i[2];
                double d;
        } xx, yy;

#endif /* ! codereview */
        unsigned s;

        /* if y is nan, replace it by x */
        if (y != y)
                y = x;

        /* if x is nan, replace it by y */
```



```
        if (x != x)
                x = y;

        /* At this point, x and y are either both numeric, or both NaN */
        if (!isnan(x) && !isgreaterequal(x, y))
                x = y;

        /*
         * clear the sign of the result if either x or y has its sign clear
         */
        xx.d = x;
        yy.d = y;
#if defined(__sparc)
        s = ~(xx.i[0] & yy.i[0]) & 0x80000000;
        xx.i[0] &= ~s;
#elif defined(__x86)
        s = ~(xx.i[1] & yy.i[1]) & 0x80000000;
        xx.i[1] &= ~s;
#else
#error Unknown architecture
#endif

        return (xx.d);
}
```

```
**********************************************************
    4302 Thu Jun  6 20:40:15 2019
new/usr/src/lib/libm/common/m9x/fmaxf.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak fmaxf = __fmaxf

/*
 * fmax(x,y) returns the larger of x and y.  If just one of the
 * arguments is NaN, fmax returns the other argument.  If both
 * arguments are NaN, fmax returns NaN (ideally, one of the
 * argument NaNs).
 *
 * C99 does not require that fmax(-0,+0) = fmax(+0,-0) = +0, but
 * ideally fmax should satisfy this.
 *
 * C99 makes no mention of exceptions for fmax.  I suppose ideally
 * either fmax never raises any exceptions or else it raises the
 * invalid operation exception if and only if some argument is a
 * signaling NaN.  In the former case, fmax should always return
 * one of its arguments.  In the latter, fmax shouldn't return a
 * signaling NaN, although when both arguments are signaling NaNs,
 * this ideal is at odds with the stipulation that fmax should
 * always return one of its arguments.
 *
 * Commutativity of fmax follows from the properties listed above
 * except when both arguments are NaN.  In that case, fmax may be
 * declared commutative by fiat because there is no portable way
 * to tell different NaNs apart.  Ideally fmax would be truly com-
 * mutative for all arguments.
 *
 * On SPARC V8, fmax must involve tests and branches.  Ideally,
 * an implementation on SPARC V9 should avoid branching, using
 * conditional moves instead where necessary, and be as efficient
 * as possible in its use of other resources.
```



```
 *
 * It appears to be impossible to attain all of the aforementioned
 * ideals simultaneously.  The implementation below satisfies the
 * following (on SPARC):
 *
 * 1. fmax(x,y) returns the larger of x and y if neither x nor y
 *    is NaN and the non-NaN argument if just one of x or y is NaN.
 *    If both x and y are NaN, fmax(x,y) returns x unchanged.
 * 2. fmax(-0,+0) = fmax(+0,-0) = +0.
 * 3. If either argument is a signaling NaN, fmax raises the invalid
 *    operation exception.  Otherwise, it raises no exceptions.
 */

#include "libm.h"                       /* for isgreaterequal macro */

float
__fmaxf(float x, float y)
{
__fmaxf(float x, float y) {
        /*
         * On SPARC v8plus/v9, this could be implemented as follows
         * (assuming %f0 = x, %f1 = y, return value left in %f0):
         *
         * fcmps        %fcc0,%f1,%f1
         * fmovsu       %fcc0,%f0,%f1
         * fcmps        %fcc0,%f0,%f1
         * fmovsul      %fcc0,%f1,%f0
         * st           %f0,[x]
         * st           %f1,[y]
         * ld           [x],%l0
         * ld           [y],%l1
         * and          %l0,%l1,%l2
         * sethi        %hi(0x80000000),%l3
         * andn         %l3,%l2,%l2
         * andn         %l0,%l2,%l0
         * st           %l0,[x]
         * ld           [x],%f0
         *
         * If VIS instructions are available, use this code instead:
         *
         * fcmps        %fcc0,%f1,%f1
         * fmovsu       %fcc0,%f0,%f1
         * fcmps        %fcc0,%f0,%f1
         * fmovsul      %fcc0,%f1,%f0
         * fands        %f0,%f1,%f2
         * fzeros       %f3
         * fnegs        %f3,%f3
         * fandnot2s %f3,%f2,%f2
         * fandnot2s %f0,%f2,%f0
         *
         * If VIS 3.0 instructions are available, use this:
         *
         * flcmps       %fcc0,%f0,%f1
         * fmovslg      %fcc0,%f1,%f0   ! move if %fcc0 is 1 or 2
         */

        union {
                unsigned i;
                float f;
        } xx, yy;

#endif /* ! codereview */
        unsigned s;

        /* if y is nan, replace it by x */
        if (y != y)
```

```
                y = x;

        /* if x is nan, replace it by y */
        if (x != x)
                x = y;

        /* At this point, x and y are either both numeric, or both NaN */
        if (!isnan(x) && !isgreaterequal(x, y))
                x = y;

        /*
         * clear the sign of the result if either x or y has its sign clear
         */
        xx.f = x;
        yy.f = y;
        s = ~(xx.i & yy.i) & 0x80000000;
        xx.i &= ~s;

        return (xx.f);
}
```

```
**********************************************************
    1852 Thu Jun  6 20:40:15 2019
new/usr/src/lib/libm/common/m9x/fmaxl.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak fmaxl = __fmaxl

#include "libm.h"                       /* for isgreaterequal macro */

long double
__fmaxl(long double x, long double y)
{
__fmaxl(long double x, long double y) {
        union {
#if defined(__sparc)
                unsigned i[4];
#elif defined(__x86)
                unsigned i[3];
#else
#error Unknown architecture
#endif

#endif /* ! codereview */
                long double ld;
        } xx, yy;

#endif /* ! codereview */
        unsigned s;

        /* if y is nan, replace it by x */
        if (y != y)
                y = x;

        /* if x is nan, replace it by y */
        if (x != x)
```



```
                x = y;

        /* At this point, x and y are either both numeric, or both NaN */
        if (!isnan(x) && !isgreaterequal(x, y))
                x = y;

        /*
         * clear the sign of the result if either x or y has its sign clear
         */
        xx.ld = x;
        yy.ld = y;
#if defined(__sparc)
        s = ~(xx.i[0] & yy.i[0]) & 0x80000000;
        xx.i[0] &= ~s;
#elif defined(__x86)
        s = ~(xx.i[2] & yy.i[2]) & 0x8000;
        xx.i[2] &= ~s;
#else
#error Unknown architecture
#endif

        return (xx.ld);
}
```

```
**********************************************************
    1982 Thu Jun  6 20:40:16 2019
new/usr/src/lib/libm/common/m9x/fmin.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak fmin = __fmin

/*
 * fmin(x,y) returns the smaller of x and y.  If just one of the
 * arguments is NaN, fmin returns the other argument.  If both
 * arguments are NaN, fmin returns NaN.
 *
 * See fmaxf.c for a discussion of implementation trade-offs.
 */

#include "libm.h"                       /* for islessequal macro */

#include "fenv_inlines.h"
#include <stdio.h>
#include <sys/isa_defs.h>

double
__fmin(double x, double y)
{
__fmin(double x, double y) {
        union {
                unsigned i[2];
                double d;
        } xx, yy;

#endif /* ! codereview */
        unsigned s;

        /* if y is nan, replace it by x */
        if (y != y)
```



```
                y = x;

        /* if x is nan, replace it by y */
        if (x != x)
                x = y;

        /* At this point, x and y are either both numeric, or both NaN */
        if (!isnan(x) && !islessequal(x, y))
                x = y;

        /*
         * set the sign of the result if either x or y has its sign set
         */
        xx.d = x;
        yy.d = y;
#if defined(_BIG_ENDIAN)
        s = (xx.i[0] | yy.i[0]) & 0x80000000;
        xx.i[0] |= s;
#else
        s = (xx.i[1] | yy.i[1]) & 0x80000000;
        xx.i[1] |= s;
#endif

        return (xx.d);
}
```

```
**********************************************************
    2390 Thu Jun  6 20:40:16 2019
new/usr/src/lib/libm/common/m9x/fminf.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak fminf = __fminf

#include "libm.h"                       /* for islessequal macro */

float
__fminf(float x, float y)
{
__fminf(float x, float y) {
        /*
         * On SPARC v8plus/v9, this could be implemented as follows
         * (assuming %f0 = x, %f1 = y, return value left in %f0):
         *
         * fcmps        %fcc0,%f1,%f1
         * fmovsu       %fcc0,%f0,%f1
         * fcmps        %fcc0,%f0,%f1
         * fmovsug      %fcc0,%f1,%f0
         * st           %f0,[x]
         * st           %f1,[y]
         * ld           [x],%l0
         * ld           [y],%l1
         * or           %l0,%l1,%l2
         * sethi        %hi(0x80000000),%l3
         * and          %l3,%l2,%l2
         * or           %l0,%l2,%l0
         * st           %l0,[x]
         * ld           [x],%f0
         *
         * If VIS instructions are available, use this code instead:
         *
         * fcmps        %fcc0,%f1,%f1
```



```
         * fmovsu       %fcc0,%f0,%f1
         * fcmps        %fcc0,%f0,%f1
         * fmovsug      %fcc0,%f1,%f0
         * fors         %f0,%f1,%f2
         * fzeros       %f3
         * fnegs        %f3,%f3
         * fands        %f3,%f2,%f2
         * fors         %f0,%f2,%f0
         *
         * If VIS 3.0 instructions are available, use this:
         *
         * flcmps       %fcc0,%f0,%f1
         * fmovsge      %fcc0,%f1,%f0   ! move if %fcc0 is 0 or 2
         */

        union {
                unsigned i;
                float f;
        } xx, yy;

#endif /* ! codereview */
        unsigned s;

        /* if y is nan, replace it by x */
        if (y != y)
                y = x;

        /* if x is nan, replace it by y */
        if (x != x)
                x = y;

        /* At this point, x and y are either both numeric, or both NaN */
        if (!isnan(x) && !islessequal(x, y))
                x = y;

        /*
         * set the sign of the result if either x or y has its sign set
         */
        xx.f = x;
        yy.f = y;
        s = (xx.i | yy.i) & 0x80000000;
        xx.i |= s;

        return (xx.f);
}
```

```
**********************************************************
    1838 Thu Jun  6 20:40:17 2019
new/usr/src/lib/libm/common/m9x/fminl.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak fminl = __fminl

#include "libm.h"                       /* for islessequal macro */

long double
__fminl(long double x, long double y)
{
__fminl(long double x, long double y) {
        union {
#if defined(__sparc)
                unsigned i[4];
#elif defined(__x86)
                unsigned i[3];
#else
#error Unknown architecture
#endif

#endif /* ! codereview */
                long double ld;
        } xx, yy;

#endif /* ! codereview */
        unsigned s;

        /* if y is nan, replace it by x */
        if (y != y)
                y = x;

        /* if x is nan, replace it by y */
        if (x != x)
```



```
                x = y;

        /* At this point, x and y are either both numeric, or both NaN */
        if (!isnan(x) && !islessequal(x, y))
                x = y;

        /*
         * set the sign of the result if either x or y has its sign set
         */
        xx.ld = x;
        yy.ld = y;
#if defined(__sparc)
        s = (xx.i[0] | yy.i[0]) & 0x80000000;
        xx.i[0] |= s;
#elif defined(__x86)
        s = (xx.i[2] | yy.i[2]) & 0x8000;
        xx.i[2] |= s;
#else
#error Unknown architecture
#endif

        return (xx.ld);
}
```

```
**********************************************************
    2806 Thu Jun  6 20:40:17 2019
new/usr/src/lib/libm/common/m9x/frexp.c
11210 libm should be cstyle(10NBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak frexp = __frexp

/*
 * frexp(x, exp) returns the normalized significand of x and sets
 * *exp so that x = r*2^(*exp) where r is the return value.  If x
 * is finite and nonzero, 1/2 <= |r| < 1.
 *
 * If x is zero, infinite or NaN, frexp returns x and sets *exp = 0.
 * (The relevant standards do not specify *exp when x is infinite or
 * NaN, but this code sets it anyway.)
 *
 * If x is a signaling NaN, this code returns x without attempting
 * to raise the invalid operation exception.  If x is subnormal,
 * this code treats it as nonzero regardless of nonstandard mode.
 */

#include "libm.h"

double
__frexp(double x, int *exp)
{
__frexp(double x, int *exp) {
        union {
                unsigned i[2];
                double d;
        } xx, yy;

#endif /* ! codereview */
        double t;
        unsigned hx;
```



```
        int e;

        xx.d = x;
        hx = xx.i[HIWORD] & ~0x80000000;

        if (hx >= 0x7ff00000) {         /* x is infinite or NaN */
                *exp = 0;
                return (x);
        }

        e = 0;

#endif /* ! codereview */
        if (hx < 0x00100000) {          /* x is subnormal or zero */
                if ((hx | xx.i[LOWORD]) == 0) {
                        *exp = 0;
                        return (x);
                }

                /*
                 * normalize x by regarding it as an integer
                 *
                 * Here we use 32-bit integer arithmetic to avoid trapping
                 * or emulating 64-bit arithmetic.  If 64-bit arithmetic is
                 * available (e.g., in SPARC V9), do this instead:
                 *
                 *  long lx = ((long) hx << 32) | xx.i[LOWORD];
                 *  xx.d = (xx.i[HIWORD] < 0)? -lx : lx;
                 *
                 * If subnormal arithmetic doesn't trap, just multiply x by
                 * a power of two.
                 */
                yy.i[HIWORD] = 0x43300000 | hx;
                yy.i[LOWORD] = xx.i[LOWORD];
                t = yy.d;
                yy.i[HIWORD] = 0x43300000;
                yy.i[LOWORD] = 0;
                t -= yy.d;              /* t = |x| scaled */
                xx.d = ((int)xx.i[HIWORD] < 0) ? -t : t;
                xx.d = ((int)xx.i[HIWORD] < 0)? -t : t;
                hx = xx.i[HIWORD] & ~0x80000000;
                e = -1074;
        }

        /* now xx.d is normal */
        xx.i[HIWORD] = (xx.i[HIWORD] & ~0x7ff00000) | 0x3fe00000;
        *exp = e + (hx >> 20) - 0x3fe;
        return (xx.d);
}
______unchanged_portion_omitted_
```

```
**********************************************************
    1640 Thu Jun  6 20:40:18 2019
new/usr/src/lib/libm/common/m9x/frexpf.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak frexpf = __frexpf

#include "libm.h"

float
__frexpf(float x, int *exp)
{
__frexpf(float x, int *exp) {
        union {
                unsigned i;
                float f;
        } xx;

#endif /* ! codereview */
        unsigned hx;
        int e;

        xx.f = x;
        hx = xx.i & ~0x80000000;

        if (hx >= 0x7f800000) {         /* x is infinite or NaN */
                *exp = 0;
                return (x);
        }

        e = 0;

#endif /* ! codereview */
        if (hx < 0x00800000) {          /* x is subnormal or zero */
                if (hx == 0) {
```



```
                        *exp = 0;
                        return (x);
                }

                /* normalize x by regarding it as an integer */
                xx.f = (int)xx.i < 0 ? -(int)hx : (int)hx;
                xx.f = (int) xx.i < 0 ? -(int) hx : (int) hx;
                hx = xx.i & ~0x80000000;
                e = -149;
        }

        /* now xx.f is normal */
        xx.i = (xx.i & ~0x7f800000) | 0x3f000000;
        *exp = e + (hx >> 23) - 0x7e;
        return (xx.f);
}
______unchanged_portion_omitted_
```

```
**********************************************************
    2631 Thu Jun  6 20:40:18 2019
new/usr/src/lib/libm/common/m9x/frexpl.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak frexpl = __frexpl

#include "libm.h"

#if defined(__sparc)

long double
__frexpl(long double x, int *exp)
{
__frexpl(long double x, int *exp) {
        union {
                unsigned i[4];
                long double q;
        } xx;

#endif /* ! codereview */
        unsigned hx;
        int e, s;

        xx.q = x;
        hx = xx.i[0] & ~0x80000000;

        if (hx >= 0x7fff0000) {         /* x is infinite or NaN */
                *exp = 0;
                return (x);
        }

        e = 0;

#endif /* ! codereview */
```



```
        if (hx < 0x00010000) {          /* x is subnormal or zero */
                if ((hx | xx.i[1] | xx.i[2] | xx.i[3]) == 0) {
                        *exp = 0;
                        return (x);
                }

                /* normalize x */
                s = xx.i[0] & 0x80000000;

#endif /* ! codereview */
                while ((hx | (xx.i[1] & 0xffff0000)) == 0) {
                        hx = xx.i[1];
                        xx.i[1] = xx.i[2];
                        xx.i[2] = xx.i[3];
                        xx.i[3] = 0;
                        e -= 32;
                }

#endif /* ! codereview */
                while (hx < 0x10000) {
                        hx = (hx << 1) | (xx.i[1] >> 31);
                        xx.i[1] = (xx.i[1] << 1) | (xx.i[2] >> 31);
                        xx.i[2] = (xx.i[2] << 1) | (xx.i[3] >> 31);
                        xx.i[3] <<= 1;
                        e--;
                }

#endif /* ! codereview */
                xx.i[0] = s | hx;
        }

        /* now xx.q is normal */
        xx.i[0] = (xx.i[0] & ~0x7fff0000) | 0x3ffe0000;
        *exp = e + (hx >> 16) - 0x3ffe;
        return (xx.q);
}

#elif defined(__x86)

long double
__frexpl(long double x, int *exp)
{
__frexpl(long double x, int *exp) {
        union {
                unsigned i[3];
                long double e;
        } xx;

#endif /* ! codereview */
        unsigned hx;
        int e;

        xx.e = x;
        hx = xx.i[2] & 0x7fff;

        if (hx >= 0x7fff) {             /* x is infinite or NaN */
                *exp = 0;
                return (x);
        }

        e = 0;

#endif /* ! codereview */
        if (hx < 0x0001) {              /* x is subnormal or zero */
                if ((xx.i[0] | xx.i[1]) == 0) {
                        *exp = 0;
```

```
                        return (x);
                }

                /* normalize x */
                xx.e *= 18446744073709551616.0L;        /* 2^64 */
                hx = xx.i[2] & 0x7fff;
                e = -64;
        }

        /* now xx.e is normal */
        xx.i[2] = (xx.i[2] & 0x8000) | 0x3ffe;
        *exp = e + hx - 0x3ffe;
        return (xx.e);
}

#else
#error Unknown architecture
#endif
```

```
**********************************************************
    1616 Thu Jun  6 20:40:19 2019
new/usr/src/lib/libm/common/m9x/ldexp.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __ldexp = ldexp

#include "libm.h"
#include <errno.h>

double
ldexp(double x, int n)
{
        int *px = (int *)&x, ix = px[HIWORD] & ~0x80000000;
ldexp(double x, int n) {
        int *px = (int *) &x, ix = px[HIWORD] & ~0x80000000;

        if (ix >= 0x7ff00000 || (px[LOWORD] | ix) == 0)
#if defined(FPADD_TRAPS_INCOMPLETE_ON_NAN)
                return (ix >= 0x7ff80000 ? x : x + x);

#endif /* ! codereview */
        /* assumes sparc-like QNaN */
#else
                return (x + x);
#endif
        x = scalbn(x, n);
        ix = px[HIWORD] & ~0x80000000;

#endif /* ! codereview */
        /*
         * SVID3 requires both overflow and underflow cases to set errno
         * XPG3/XPG4/XPG4.2/SUSv2 requires overflow to set errno
         */
        if (ix >= 0x7ff00000 || (px[LOWORD] | ix) == 0)
```



```
                errno = ERANGE;

#endif /* ! codereview */
        return (x);
}
```

```
**********************************************************
    1107 Thu Jun  6 20:40:19 2019
new/usr/src/lib/libm/common/m9x/ldexpf.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __ldexpf = ldexpf

#include "libm.h"

float
ldexpf(float x, int n)
{
ldexpf(float x, int n) {
        return (scalbnf(x, n));
}
______unchanged_portion_omitted_
```

```
**********************************************************
    1119 Thu Jun  6 20:40:20 2019
new/usr/src/lib/libm/common/m9x/ldexpl.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __ldexpl = ldexpl

#include "libm.h"

long double
ldexpl(long double x, int n)
{
ldexpl(long double x, int n) {
        return (scalbnl(x, n));
}
______unchanged_portion_omitted_
```

```
**********************************************************
    2284 Thu Jun  6 20:40:20 2019
new/usr/src/lib/libm/common/m9x/llrint.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __llrint = llrint
#if defined(__sparcv9) || defined(__amd64)
#pragma weak lrint = llrint
#pragma weak __lrint = llrint
#endif

/*
 * llrint(x) rounds its argument to the nearest integer according
 * to the current rounding direction and converts the result to a
 * 64 bit signed integer.
 *
 * If x is NaN, infinite, or so large that the nearest integer would
 * exceed 64 bits, the invalid operation exception is raised.  If x
 * is not an integer, the inexact exception is raised.
 */

#include "libm.h"

long long
llrint(double x)
{
llrint(double x) {
        /*
         * Note: The following code works on x86 (in the default rounding
         * precision mode), but one should just use the fistpll instruction
         * instead.
         */
        union {
                unsigned i[2];
                double d;
```



```
        } xx, yy;

#endif /* ! codereview */
        unsigned hx;

        xx.d = x;
        hx = xx.i[HIWORD] & ~0x80000000;

        if (hx < 0x43300000) {          /* |x| < 2^52 */
                /* add and subtract a power of two to round x to an integer */
#if defined(__sparc) || defined(__amd64)
                yy.i[HIWORD] = (xx.i[HIWORD] & 0x80000000) | 0x43300000;
#elif defined(__i386)                   /* !defined(__amd64) */
                yy.i[HIWORD] = (xx.i[HIWORD] & 0x80000000) | 0x43e00000;
#else
#error Unknown architecture
#endif
                yy.i[LOWORD] = 0;
                x = (x + yy.d) - yy.d;
        }

        /* now x is nan, inf, or integral */
        return ((long long)x);
        return ((long long) x);
}
______unchanged_portion_omitted_
```

```
**********************************************************
    1997 Thu Jun  6 20:40:21 2019
new/usr/src/lib/libm/common/m9x/llrintf.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __llrintf = llrintf
#if defined(__sparcv9) || defined(__amd64)
#pragma weak lrintf = llrintf
#pragma weak __lrintf = llrintf
#endif

#include "libm.h"

long long
llrintf(float x)
{
llrintf(float x) {
        /*
         * Note: The following code works on x86 (in the default rounding
         * precision mode), but one should just use the fistpll instruction
         * instead.
         */
        union {
                unsigned i;
                float f;
        } xx, yy;

#endif /* ! codereview */
        unsigned hx;

        xx.f = x;
        hx = xx.i & ~0x80000000;

        if (hx < 0x4b000000) {          /* |x| < 2^23 */
                /* add and subtract a power of two to round x to an integer */
```



```
#if defined(__sparc) || defined(__amd64)
                yy.i = (xx.i & 0x80000000) | 0x4b000000;
#elif defined(__i386)
                /* assume 64-bit precision */
                yy.i = (xx.i & 0x80000000) | 0x5f000000;
#else
#error Unknown architecture
#endif
                x = (x + yy.f) - yy.f;

                /*
                 * on LP32 architectures, we can just convert x to a 32-bit
                 * integer and sign-extend it
                 */
                return ((long)x);
                return ((long) x);
        }

        /* now x is nan, inf, or integral */
        return ((long long)x);
        return ((long long) x);
}
______unchanged_portion_omitted_
```

```
**********************************************************
    4336 Thu Jun  6 20:40:21 2019
new/usr/src/lib/libm/common/m9x/llrintl.c
11210 libm should be cstyle(10NBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __llrintl = llrintl
#if defined(__sparcv9) || defined(__amd64)
#pragma weak lrintl = llrintl
#pragma weak __lrintl = llrintl
#endif

#include "libm.h"

#if defined(__sparc)

#include "fma.h"
#include "fenv_inlines.h"

long long
llrintl(long double x)
{
llrintl(long double x) {
        union {
                unsigned i[4];
                long double q;
        } xx;
        union {
                unsigned i[2];
                long long l;
        } zz;
        union {
                unsigned i;
                float f;
        } tt;
```



```
#endif /* ! codereview */
        unsigned int hx, sx, frac, fsr;
        int rm, j;
        volatile float dummy;

        xx.q = x;
        sx = xx.i[0] & 0x80000000;
        hx = xx.i[0] & ~0x80000000;

        /* handle trivial cases */
        if (hx > 0x403e0000) {          /* |x| > 2^63 + ... or x is nan */
                /* convert an out-of-range float */
                tt.i = sx | 0x7f000000;
                return ((long long)tt.f);
        } else if ((hx | xx.i[1] | xx.i[2] | xx.i[3]) == 0) {   /* x is zero */
                return ((long long) tt.f);
        } else if ((hx | xx.i[1] | xx.i[2] | xx.i[3]) == 0) /* x is zero */
                return (0LL);
        }
#endif /* ! codereview */

        /* get the rounding mode */
        __fenv_getfsr32(&fsr);
        rm = fsr >> 30;

        /* flip the sense of directed roundings if x is negative */
        if (sx)
                rm ^= rm >> 1;

        /* handle |x| < 1 */
        if (hx < 0x3fff0000) {
                dummy = 1.0e30f;        /* x is nonzero, so raise inexact */
                dummy += 1.0e-30f;

                if (rm == FSR_RP || (rm == FSR_RN && (hx >= 0x3ffe0000 && ((hx &
                    0xffff) | xx.i[1] | xx.i[2] | xx.i[3]))))
                if (rm == FSR_RP || (rm == FSR_RN && (hx >= 0x3ffe0000 &&
                        ((hx & 0xffff) | xx.i[1] | xx.i[2] | xx.i[3]))))
                        return (sx ? -1LL : 1LL);

#endif /* ! codereview */
                return (0LL);
        }

        /* extract the integer and fractional parts of x */
        j = 0x406f - (hx >> 16);
        xx.i[0] = 0x10000 | (xx.i[0] & 0xffff);

#endif /* ! codereview */
        if (j >= 96) {
                zz.i[0] = 0;
                zz.i[1] = xx.i[0] >> (j - 96);
                frac = ((xx.i[0] << 1) << (127 - j)) | (xx.i[1] >> (j - 96));

#endif /* ! codereview */
                if (((xx.i[1] << 1) << (127 - j)) | xx.i[2] | xx.i[3])
                        frac |= 1;
        } else if (j >= 64) {
                zz.i[0] = xx.i[0] >> (j - 64);
                zz.i[1] = ((xx.i[0] << 1) << (95 - j)) | (xx.i[1] >> (j - 64));
                frac = ((xx.i[1] << 1) << (95 - j)) | (xx.i[2] >> (j - 64));

#endif /* ! codereview */
                if (((xx.i[2] << 1) << (95 - j)) | xx.i[3])
                        frac |= 1;
        } else {
```

```
                zz.i[0] = ((xx.i[0] << 1) << (63 - j)) | (xx.i[1] >> (j - 32));
                zz.i[1] = ((xx.i[1] << 1) << (63 - j)) | (xx.i[2] >> (j - 32));
                frac = ((xx.i[2] << 1) << (63 - j)) | (xx.i[3] >> (j - 32));

#endif /* ! codereview */
                if ((xx.i[3] << 1) << (63 - j))
                        frac |= 1;
        }

        /* round */
        if (frac && (rm == FSR_RP || (rm == FSR_RN && (frac > 0x80000000u ||
            (frac == 0x80000000 && (zz.i[1] & 1))))) {
                if (++zz.i[1] == 0)
                        zz.i[0]++;
        }

        /* check for result out of range (note that z is |x| at this point) */
        if (zz.i[0] > 0x80000000u || (zz.i[0] == 0x80000000 && (zz.i[1] ||
            !sx))) {
                tt.i = sx | 0x7f000000;
                return ((long long)tt.f);
                return ((long long) tt.f);
        }

        /* raise inexact if need be */
        if (frac) {
                dummy = 1.0e30F;
                dummy += 1.0e-30F;
        }

        /* negate result if need be */
        if (sx) {
                zz.i[0] = ~zz.i[0];
                zz.i[1] = -zz.i[1];

#endif /* ! codereview */
                if (zz.i[1] == 0)
                        zz.i[0]++;
        }

#endif /* ! codereview */
        return (zz.l);
}
#elif defined(__x86)
long long
llrintl(long double x)
{
llrintl(long double x) {
        /*
         * Note: The following code works on x86 (in the default rounding
         * precision mode), but one ought to just use the fistpll instruction
         * instead.
         */
        union {
                unsigned i[3];
                long double e;
        } xx, yy;

#endif /* ! codereview */
        int ex;

        xx.e = x;
        ex = xx.i[2] & 0x7fff;

        if (ex < 0x403e) {              /* |x| < 2^63 */
                /* add and subtract a power of two to round x to an integer */
```



```
                yy.i[2] = (xx.i[2] & 0x8000) | 0x403e;
                yy.i[1] = 0x80000000;
                yy.i[0] = 0;
                x = (x + yy.e) - yy.e;
        }

        /* now x is nan, inf, or integral */
        return ((long long)x);
        return ((long long) x);
}
_____unchanged_portion_omitted_
```

```
**********************************************************
    2190 Thu Jun  6 20:40:22 2019
new/usr/src/lib/libm/common/m9x/llround.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __llround = llround
#if defined(__sparcv9) || defined(__amd64)
#pragma weak lround = llround
#pragma weak __lround = llround
#endif

/*
 * llround(x) rounds its argument to the nearest integer, rounding
 * ties away from zero, and converts the result to a 64 bit signed
 * integer.
 *
 * If x is NaN, infinite, or so large that the nearest integer
 * would exceed 64 bits, the invalid operation exception is raised.
 */

#include "libm.h"

long long
llround(double x)
{
llround(double x) {
        union {
                unsigned i[2];
                double d;
        } xx;

#endif /* ! codereview */
        unsigned hx, sx, i;

        xx.d = x;
```



```
        hx = xx.i[HIWORD] & ~0x80000000;
        sx = xx.i[HIWORD] & 0x80000000;

        if (hx < 0x43300000) {          /* |x| < 2^52 */
                /* handle |x| < 1 */
                if (hx < 0x3ff00000) {
                        if (hx >= 0x3fe00000)
                                return (sx ? -1LL : 1LL);

#endif /* ! codereview */
                        return (0LL);
                }

                /* round x at the integer bit */
                if (hx < 0x41300000) {
                        i = 1 << (0x412 - (hx >> 20));
                        xx.i[HIWORD] = (xx.i[HIWORD] + i) & ~(i | (i - 1));
                        xx.i[LOWORD] = 0;
                } else {
                        i = 1 << (0x432 - (hx >> 20));
                        xx.i[LOWORD] += i;

#endif /* ! codereview */
                        if (xx.i[LOWORD] < i)
                                xx.i[HIWORD]++;

#endif /* ! codereview */
                        xx.i[LOWORD] &= ~(i | (i - 1));
                }
        }

        /* now x is nan, inf, or integral */
        return ((long long)xx.d);
        return ((long long) xx.d);
}
_______unchanged_portion_omitted_
```

```
**********************************************************
    1797 Thu Jun  6 20:40:22 2019
new/usr/src/lib/libm/common/m9x/llroundf.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __llroundf = llroundf
#if defined(__sparcv9) || defined(__amd64)
#pragma weak lroundf = llroundf
#pragma weak __lroundf = llroundf
#endif

#include "libm.h"

long long
llroundf(float x)
{
llroundf(float x) {
        union {
                unsigned i;
                float f;
        } xx;

#endif /* ! codereview */
        unsigned hx, sx, i;

        xx.f = x;
        hx = xx.i & ~0x80000000;
        sx = xx.i & 0x80000000;

        if (hx < 0x4b000000) {          /* |x| < 2^23 */
                /* handle |x| < 1 */
                if (hx < 0x3f800000) {
                        if (hx >= 0x3f000000)
                                return (sx ? -1LL : 1LL);
```



```
#endif /* ! codereview */
                        return (0LL);
                }

                /* round x at the integer bit */
                i = 1 << (0x95 - (hx >> 23));
                xx.i = (xx.i + i) & ~((i << 1) - 1);

                /*
                 * on LP32 architectures, we can just convert x to a 32-bit
                 * integer and sign-extend it
                 */
                return ((long)xx.f);
                return ((long) xx.f);
        }

        /* now x is nan, inf, or integral */
        return ((long long)x);
        return ((long long) x);
}
______unchanged_portion_omitted_
```

```
**********************************************************
    3834 Thu Jun  6 20:40:23 2019
new/usr/src/lib/libm/common/m9x/llroundl.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __llroundl = llroundl
#if defined(__sparcv9) || defined(__amd64)
#pragma weak lroundl = llroundl
#pragma weak __lroundl = llroundl
#endif

#include "libm.h"

#if defined(__sparc)
long long
llroundl(long double x)
{
llroundl(long double x) {
        union {
                unsigned i[4];
                long double q;
        } xx;
        union {
                unsigned i[2];
                long long l;
        } zz;
        union {
                unsigned i;
                float f;
        } tt;

#endif /* ! codereview */
        unsigned hx, sx, frac;
        int j;
```



```
        xx.q = x;
        sx = xx.i[0] & 0x80000000;
        hx = xx.i[0] & ~0x80000000;

        /* handle trivial cases */
        if (hx > 0x403e0000) {          /* |x| > 2^63 + ... or x is nan */
                /* convert an out-of-range float */
                tt.i = sx | 0x7f000000;
                return ((long long)tt.f);
                return ((long long) tt.f);
        }

        /* handle |x| < 1 */
        if (hx < 0x3fff0000) {
                if (hx >= 0x3ffe0000)
                        return (sx ? -1LL : 1LL);

#endif /* ! codereview */
                return (0LL);
        }

        /* extract the integer and fractional parts of x */
        j = 0x406f - (hx >> 16);
        xx.i[0] = 0x10000 | (xx.i[0] & 0xffff);

#endif /* ! codereview */
        if (j >= 96) {
                zz.i[0] = 0;
                zz.i[1] = xx.i[0] >> (j - 96);
                frac = ((xx.i[0] << 1) << (127 - j)) | (xx.i[1] >> (j - 96));

#endif /* ! codereview */
                if (((xx.i[1] << 1) << (127 - j)) | xx.i[2] | xx.i[3])
                        frac |= 1;
        } else if (j >= 64) {
                zz.i[0] = xx.i[0] >> (j - 64);
                zz.i[1] = ((xx.i[0] << 1) << (95 - j)) | (xx.i[1] >> (j - 64));
                frac = ((xx.i[1] << 1) << (95 - j)) | (xx.i[2] >> (j - 64));

#endif /* ! codereview */
                if (((xx.i[2] << 1) << (95 - j)) | xx.i[3])
                        frac |= 1;
        } else {
                zz.i[0] = ((xx.i[0] << 1) << (63 - j)) | (xx.i[1] >> (j - 32));
                zz.i[1] = ((xx.i[1] << 1) << (63 - j)) | (xx.i[2] >> (j - 32));
                frac = ((xx.i[2] << 1) << (63 - j)) | (xx.i[3] >> (j - 32));

#endif /* ! codereview */
                if ((xx.i[3] << 1) << (63 - j))
                        frac |= 1;
        }

        /* round */
        if (frac >= 0x80000000u) {
                if (++zz.i[1] == 0)
                        zz.i[0]++;
        }

        /* check for result out of range (note that z is |x| at this point) */
        if (zz.i[0] > 0x80000000u || (zz.i[0] == 0x80000000 && (zz.i[1] ||
            !sx))) {
                tt.i = sx | 0x7f000000;
                return ((long long)tt.f);
                return ((long long) tt.f);
        }
```

```
        /* negate result if need be */
        if (sx) {
                zz.i[0] = ~zz.i[0];
                zz.i[1] = -zz.i[1];
#endif /* ! codereview */
                if (zz.i[1] == 0)
                        zz.i[0]++;
        }

        return (zz.l);
}
#elif defined(__x86)
long long
llroundl(long double x)
{
llroundl(long double x) {
        union {
                unsigned i[3];
                long double e;
        } xx;
#endif /* ! codereview */
        int ex, sx, i;

        xx.e = x;
        ex = xx.i[2] & 0x7fff;
        sx = xx.i[2] & 0x8000;

        if (ex < 0x403e) {              /* |x| < 2^63 */
                /* handle |x| < 1 */
                if (ex < 0x3fff) {
                        if (ex >= 0x3ffe)
                                return (sx ? -1LL : 1LL);
#endif /* ! codereview */
                        return (0LL);
                }

                /* round x at the integer bit */
                if (ex < 0x401e) {
                        i = 1 << (0x401d - ex);
                        xx.i[1] = (xx.i[1] + i) & ~(i | (i - 1));
                        xx.i[0] = 0;
                } else {
                        i = 1 << (0x403d - ex);
                        xx.i[0] += i;
#endif /* ! codereview */
                        if (xx.i[0] < i)
                                xx.i[1]++;
#endif /* ! codereview */
                        xx.i[0] &= ~(i | (i - 1));
                }
#endif /* ! codereview */
                if (xx.i[1] == 0) {
                        xx.i[2] = sx | ++ex;
                        xx.i[1] = 0x80000000U;
                }
        }

        /* now x is nan, inf, or integral */
        return ((long long)xx.e);
        return ((long long) xx.e);
```



```
}
```

_____*unchanged_portion_omitted_*

```
**********************************************************
    2240 Thu Jun  6 20:40:23 2019
new/usr/src/lib/libm/common/m9x/lrint.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __lrint = lrint

/*
 * lrint(x) rounds its argument to the nearest integer according
 * to the current rounding direction and converts the result to
 * a 32 bit signed integer.
 *
 * If x is NaN, infinite, or so large that the nearest integer
 * would exceed 32 bits, the invalid operation exception is raised.
 * If x is not an integer, the inexact exception is raised.
 */

#include <sys/isa_defs.h>               /* _ILP32 */
#include "libm.h"

#if defined(_ILP32)
long
lrint(double x)
{
lrint(double x) {
        /*
         * Note: The following code works on x86 (in the default rounding
         * precision mode), but one should just use the fistpl instruction
         * instead.
         */
        union {
                unsigned i[2];
                double d;
        } xx, yy;
```



```
#endif /* ! codereview */
        unsigned hx;

        xx.d = x;
        hx = xx.i[HIWORD] & ~0x80000000;

#endif /* ! codereview */
        if (hx < 0x43300000) {          /* |x| < 2^52 */
                /* add and subtract a power of two to round x to an integer */
#if defined(__sparc)
                yy.i[HIWORD] = (xx.i[HIWORD] & 0x80000000) | 0x43300000;
#elif defined(__x86)
                yy.i[HIWORD] = (xx.i[HIWORD] & 0x80000000) | 0x43e00000;
#else
#error Unknown architecture
#endif
                yy.i[LOWORD] = 0;
                x = (x + yy.d) - yy.d;
        }

        /* now x is nan, inf, or integral */
        return ((long)x);
        return ((long) x);
}
______unchanged_portion_omitted_
```

```
**********************************************************
    1871 Thu Jun  6 20:40:24 2019
new/usr/src/lib/libm/common/m9x/lrintf.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __lrintf = lrintf

#include <sys/isa_defs.h>               /* _ILP32 */
#include "libm.h"

#if defined(_ILP32)
long
lrintf(float x)
{
lrintf(float x) {
        /*
         * Note: The following code works on x86 (in the default rounding
         * precision mode), but one should just use the fistpl instruction
         * instead.
         */
        union {
                unsigned i;
                float f;
        } xx, yy;

#endif /* ! codereview */
        unsigned hx;

        xx.f = x;
        hx = xx.i & ~0x80000000;

#endif /* ! codereview */
        if (hx < 0x4b000000) {          /* |x| < 2^23 */
                /* add and subtract a power of two to round x to an integer */
#if defined(__sparc)
```



```
                yy.i = (xx.i & 0x80000000) | 0x4b000000;
#elif defined(__x86)
                /* assume 64-bit precision */
                yy.i = (xx.i & 0x80000000) | 0x5f000000;
#else
#error Unknown architecture
#endif
                x = (x + yy.f) - yy.f;
                return ((long)x);
                return ((long) x);
        }

        /* now x is nan, inf, or integral */
        return ((long)x);
        return ((long) x);
}
______unchanged_portion_omitted_
```

```
**********************************************************
    3899 Thu Jun  6 20:40:24 2019
new/usr/src/lib/libm/common/m9x/lrintl.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __lrintl = lrintl

#include <sys/isa_defs.h>               /* _ILP32 */
#include "libm.h"

#if defined(_ILP32)
#if defined(__sparc)

#include "fma.h"
#include "fenv_inlines.h"

long
lrintl(long double x)
{
lrintl(long double x) {
        union {
                unsigned int i[4];
                long double q;
        } xx;
        union {
                unsigned int i;
                float f;
        } tt;

#endif /* ! codereview */
        unsigned int hx, sx, frac, l, fsr;
        int rm, j;
        volatile float dummy;

        xx.q = x;
```



```
        sx = xx.i[0] & 0x80000000;
        hx = xx.i[0] & ~0x80000000;

        /* handle trivial cases */
        if (hx > 0x401e0000) {          /* |x| > 2^31 + ... or x is nan */
                /* convert an out-of-range float */
                tt.i = sx | 0x7f000000;
                return ((long)tt.f);
        } else if ((hx | xx.i[1] | xx.i[2] | xx.i[3]) == 0) {   /* x is zero */
                return ((long) tt.f);
        } else if ((hx | xx.i[1] | xx.i[2] | xx.i[3]) == 0) /* x is zero */
                return (0L);
        }
#endif /* ! codereview */

        /* get the rounding mode */
        __fenv_getfsr32(&fsr);
        rm = fsr >> 30;

        /* flip the sense of directed roundings if x is negative */
        if (sx)
                rm ^= rm >> 1;

        /* handle |x| < 1 */
        if (hx < 0x3fff0000) {
                dummy = 1.0e30F;        /* x is nonzero, so raise inexact */
                dummy += 1.0e-30F;

                if (rm == FSR_RP || (rm == FSR_RN && (hx >= 0x3ffe0000 && ((hx &
                    0xffff) | xx.i[1] | xx.i[2] | xx.i[3]))))
                if (rm == FSR_RP || (rm == FSR_RN && (hx >= 0x3ffe0000 &&
                        ((hx & 0xffff) | xx.i[1] | xx.i[2] | xx.i[3]))))
                        return (sx ? -1L : 1L);

#endif /* ! codereview */
                return (0L);
        }

        /* extract the integer and fractional parts of x */
        j = 0x406f - (hx >> 16);        /* 91 <= j <= 112 */
        xx.i[0] = 0x10000 | (xx.i[0] & 0xffff);

#endif /* ! codereview */
        if (j >= 96) {                  /* 96 <= j <= 112 */
                l = xx.i[0] >> (j - 96);
                frac = ((xx.i[0] << 1) << (127 - j)) | (xx.i[1] >> (j - 96));

#endif /* ! codereview */
                if (((xx.i[1] << 1) << (127 - j)) | xx.i[2] | xx.i[3])
                        frac |= 1;
        } else {                        /* 91 <= j <= 95 */
                l = (xx.i[0] << (96 - j)) | (xx.i[1] >> (j - 64));
                frac = (xx.i[1] << (96 - j)) | (xx.i[2] >> (j - 64));

#endif /* ! codereview */
                if ((xx.i[2] << (96 - j)) | xx.i[3])
                        frac |= 1;
        }

        /* round */
        if (frac && (rm == FSR_RP || (rm == FSR_RN && (frac > 0x80000000U ||
            (frac == 0x80000000 && (l & 1))))))
                l++;

        /* check for result out of range (note that z is |x| at this point) */
        if (l > 0x80000000U || (l == 0x80000000U && !sx)) {
```

```
122                 tt.i = sx | 0x7f000000;
123                 return ((long)tt.f);
 45                 return ((long) tt.f);
124         }

126         /* raise inexact if need be */
127         if (frac) {
128                 dummy = 1.0e30F;
129                 dummy += 1.0e-30F;
130         }

132         /* negate result if need be */
133         if (sx)
134                 l = -l;

136         return ((long)l);
 57         return ((long) l);
137 }
138 #elif defined(__x86)
139 long
140 lrintl(long double x)
141 {
 61 lrintl(long double x) {
142         /*
143          * Note: The following code works on x86 (in the default rounding
144          * precision mode), but one ought to just use the fistpl instruction
145          * instead.
146          */
147         union {
148                 unsigned i[3];
149                 long double e;
150         } xx, yy;

152 #endif /* ! codereview */
153         int ex;

155         xx.e = x;
156         ex = xx.i[2] & 0x7fff;

158 #endif /* ! codereview */
159         if (ex < 0x403e) {              /* |x| < 2^63 */
160                 /* add and subtract a power of two to round x to an integer */
161                 yy.i[2] = (xx.i[2] & 0x8000) | 0x403e;
162                 yy.i[1] = 0x80000000;
163                 yy.i[0] = 0;
164                 x = (x + yy.e) - yy.e;
165         }

167         /* now x is nan, inf, or integral */
168         return ((long)x);
 71         return ((long) x);
169 }
```

_____*unchanged_portion_omitted_*

```
**********************************************************
    2176 Thu Jun  6 20:40:25 2019
new/usr/src/lib/libm/common/m9x/lround.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __lround = lround

/*
 * lround(x) rounds its argument to the nearest integer, rounding ties
 * away from zero, and converts the result to a 32 bit signed integer.
 *
 * If x is NaN, infinite, or so large that the nearest integer
 * would exceed 32 bits, the invalid operation exception is raised.
 */

#include <sys/isa_defs.h>               /* _ILP32 */
#include "libm.h"

#if defined(_ILP32)
long
lround(double x)
{
lround(double x) {
        union {
                unsigned i[2];
                double d;
        } xx;

#endif /* ! codereview */
        unsigned hx, sx, i;

        xx.d = x;
        hx = xx.i[HIWORD] & ~0x80000000;
        sx = xx.i[HIWORD] & 0x80000000;
```



```
#endif /* ! codereview */
        if (hx < 0x43300000) {          /* |x| < 2^52 */
                if (hx < 0x3ff00000) {  /* |x| < 1 */
                        if (hx >= 0x3fe00000)
                                return (sx ? -1L : 1L);

#endif /* ! codereview */
                        return (0L);
                }

                /* round x at the integer bit */
                if (hx < 0x41300000) {
                        i = 1 << (0x412 - (hx >> 20));
                        xx.i[HIWORD] = (xx.i[HIWORD] + i) & ~(i | (i - 1));
                        xx.i[LOWORD] = 0;
                } else {
                        i = 1 << (0x432 - (hx >> 20));
                        xx.i[LOWORD] += i;

#endif /* ! codereview */
                        if (xx.i[LOWORD] < i)
                                xx.i[HIWORD]++;

#endif /* ! codereview */
                        xx.i[LOWORD] &= ~(i | (i - 1));
                }
        }

        /* now x is nan, inf, or integral */
        return ((long)xx.d);
        return ((long) xx.d);
}
```

_____*unchanged_portion_omitted_*

```
**********************************************************
    1677 Thu Jun  6 20:40:25 2019
new/usr/src/lib/libm/common/m9x/lroundf.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __lroundf = lroundf

#include <sys/isa_defs.h>               /* _ILP32 */
#include "libm.h"

#if defined(_ILP32)
long
lroundf(float x)
{
lroundf(float x) {
        union {
                unsigned i;
                float f;
        } xx;

#endif /* ! codereview */
        unsigned hx, sx, i;

        xx.f = x;
        hx = xx.i & ~0x80000000;
        sx = xx.i & 0x80000000;

#endif /* ! codereview */
        if (hx < 0x4b000000) {          /* |x| < 2^23 */
                if (hx < 0x3f800000) {  /* |x| < 1 */
                        if (hx >= 0x3f000000)
                                return (sx ? -1L : 1L);

#endif /* ! codereview */
                        return (0L);
```



```
                }

                /* round x at the integer bit */
                i = 1 << (0x95 - (hx >> 23));
                xx.i = (xx.i + i) & ~((i << 1) - 1);
                return ((long)xx.f);
                return ((long) xx.f);
        }

        /* now x is nan, inf, or integral */
        return ((long)x);
        return ((long) x);
}
______unchanged_portion_omitted_
```

```
**********************************************************
    3381 Thu Jun  6 20:40:26 2019
new/usr/src/lib/libm/common/m9x/lroundl.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __lroundl = lroundl

#include <sys/isa_defs.h>               /* _ILP32 */
#include "libm.h"

#if defined(_ILP32)
#if defined(__sparc)
long
lroundl(long double x)
{
lroundl(long double x) {
        union {
                unsigned i[4];
                long double q;
        } xx;
        union {
                unsigned i;
                float f;
        } tt;

#endif /* ! codereview */
        unsigned hx, sx, frac, l;
        int j;

        xx.q = x;
        sx = xx.i[0] & 0x80000000;
        hx = xx.i[0] & ~0x80000000;

        /* handle trivial cases */
        if (hx > 0x401e0000) {          /* |x| > 2^31 + ... or x is nan */
```



```
                /* convert an out-of-range float */
                tt.i = sx | 0x7f000000;
                return ((long)tt.f);
                return ((long) tt.f);
        }

        /* handle |x| < 1 */
        if (hx < 0x3fff0000) {
                if (hx >= 0x3ffe0000)
                        return (sx ? -1L : 1L);

#endif /* ! codereview */
                return (0L);
        }

        /* extract the integer and fractional parts of x */
        j = 0x406f - (hx >> 16);        /* 91 <= j <= 112 */
        xx.i[0] = 0x10000 | (xx.i[0] & 0xffff);

#endif /* ! codereview */
        if (j >= 96) {                  /* 96 <= j <= 112 */
                l = xx.i[0] >> (j - 96);
                frac = ((xx.i[0] << 1) << (127 - j)) | (xx.i[1] >> (j - 96));

#endif /* ! codereview */
                if (((xx.i[1] << 1) << (127 - j)) | xx.i[2] | xx.i[3])
                        frac |= 1;
        } else {                        /* 91 <= j <= 95 */
                l = (xx.i[0] << (96 - j)) | (xx.i[1] >> (j - 64));
                frac = (xx.i[1] << (96 - j)) | (xx.i[2] >> (j - 64));

#endif /* ! codereview */
                if ((xx.i[2] << (96 - j)) | xx.i[3])
                        frac |= 1;
        }

        /* round */
        if (frac >= 0x80000000U)
                l++;

        /* check for result out of range (note that z is |x| at this point) */
        if (l > 0x80000000U || (l == 0x80000000U && !sx)) {
                tt.i = sx | 0x7f000000;
                return ((long)tt.f);
                return ((long) tt.f);
        }

        /* negate result if need be */
        if (sx)
                l = -l;

        return ((long)l);
        return ((long) l);
}
#elif defined(__x86)
long
lroundl(long double x)
{
lroundl(long double x) {
        union {
                unsigned i[3];
                long double e;
        } xx;

#endif /* ! codereview */
        int ex, sx, i;
```

```
        xx.e = x;
        ex = xx.i[2] & 0x7fff;
        sx = xx.i[2] & 0x8000;

#endif /* ! codereview */
        if (ex < 0x403e) {              /* |x| < 2^63 */
                if (ex < 0x3fff) {      /* |x| < 1 */
                        if (ex >= 0x3ffe)
                                return (sx ? -1L : 1L);

#endif /* ! codereview */
                        return (0L);
                }

                /* round x at the integer bit */
                if (ex < 0x401e) {
                        i = 1 << (0x401d - ex);
                        xx.i[1] = (xx.i[1] + i) & ~(i | (i - 1));
                        xx.i[0] = 0;
                } else {
                        i = 1 << (0x403d - ex);
                        xx.i[0] += i;

#endif /* ! codereview */
                        if (xx.i[0] < i)
                                xx.i[1]++;

#endif /* ! codereview */
                        xx.i[0] &= ~(i | (i - 1));
                }

#endif /* ! codereview */
                if (xx.i[1] == 0) {
                        xx.i[2] = sx | ++ex;
                        xx.i[1] = 0x80000000U;
                }
        }

        /* now x is nan, inf, or integral */
        return ((long)xx.e);
        return ((long) xx.e);
}
```

*_____unchanged_portion_omitted_*

```
**********************************************************
    2425 Thu Jun  6 20:40:26 2019
new/usr/src/lib/libm/common/m9x/modf.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak modf = __modf
#pragma weak _modf = __modf

/*
 * modf(x, iptr) decomposes x into an integral part and a fractional
 * part both having the same sign as x.  It stores the integral part
 * in *iptr and returns the fractional part.
 *
 * If x is infinite, modf sets *iptr to x and returns copysign(0.0,x).
 * If x is NaN, modf sets *iptr to x and returns x.
 *
 * If x is a signaling NaN, this code does not attempt to raise the
 * invalid operation exception.
 */

#include "libm.h"

double
__modf(double x, double *iptr)
{
__modf(double x, double *iptr) {
        union {
                unsigned i[2];
                double d;
        } xx, yy;

#endif /* ! codereview */
        unsigned hx, s;

        xx.d = x;
```



```
        hx = xx.i[HIWORD] & ~0x80000000;

        if (hx >= 0x43300000) {         /* x is NaN, infinite, or integral */
                *iptr = x;

                if (hx < 0x7ff00000 || (hx == 0x7ff00000 && xx.i[LOWORD] ==
                    0)) {
                if (hx < 0x7ff00000 || (hx == 0x7ff00000 &&
                        xx.i[LOWORD] == 0)) {
                        xx.i[HIWORD] &= 0x80000000;
                        xx.i[LOWORD] = 0;
                }

#endif /* ! codereview */
                return (xx.d);
        }

        if (hx < 0x3ff00000) {          /* |x| < 1 */
                xx.i[HIWORD] &= 0x80000000;
                xx.i[LOWORD] = 0;
                *iptr = xx.d;
                return (x);
        }

        /* split x at the binary point */
        s = xx.i[HIWORD] & 0x80000000;

#endif /* ! codereview */
        if (hx < 0x41400000) {
                yy.i[HIWORD] = xx.i[HIWORD] & ~((1 << (0x413 - (hx >> 20))) -
                    1);
                yy.i[LOWORD] = 0;
        } else {
                yy.i[HIWORD] = xx.i[HIWORD];
                yy.i[LOWORD] = xx.i[LOWORD] & ~((1 << (0x433 - (hx >> 20))) -
                    1);
        }

#endif /* ! codereview */
        *iptr = yy.d;
        xx.d -= yy.d;
        xx.i[HIWORD] = (xx.i[HIWORD] & ~0x80000000) | s;
        /* keep sign of x */
        return (xx.d);
}
```

```
**********************************************************
    1694 Thu Jun  6 20:40:27 2019
new/usr/src/lib/libm/common/m9x/modff.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak modff = __modff
#pragma weak _modff = __modff

#include "libm.h"

float
__modff(float x, float *iptr)
{
__modff(float x, float *iptr) {
        union {
                unsigned i;
                float f;
        } xx, yy;

#endif /* ! codereview */
        unsigned hx, s;

        xx.f = x;
        hx = xx.i & ~0x80000000;

        if (hx >= 0x4b000000) {         /* x is NaN, infinite, or integral */
                *iptr = x;

#endif /* ! codereview */
                if (hx <= 0x7f800000)
                        xx.i &= 0x80000000;

#endif /* ! codereview */
                return (xx.f);
        }
```



```
        if (hx < 0x3f800000) {          /* |x| < 1 */
                xx.i &= 0x80000000;
                *iptr = xx.f;
                return (x);
        }

        /* split x at the binary point */
        s = xx.i & 0x80000000;
        yy.i = xx.i & ~((1 << (0x96 - (hx >> 23))) - 1);
        *iptr = yy.f;
        xx.f -= yy.f;
        xx.i = (xx.i & ~0x80000000) | s;
        /* restore sign in case difference is 0 */
        return (xx.f);
}
```

```
**********************************************************
    3549 Thu Jun  6 20:40:27 2019
new/usr/src/lib/libm/common/m9x/modfl.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak modfl = __modfl

#include "libm.h"

#if defined(__sparc)

long double
__modfl(long double x, long double *iptr)
{
__modfl(long double x, long double *iptr) {
        union {
                unsigned i[4];
                long double q;
        } xx, yy;

#endif /* ! codereview */
        unsigned hx, s;

        xx.q = x;
        hx = xx.i[0] & ~0x80000000;

        if (hx >= 0x406f0000) {         /* x is NaN, infinite, or integral */
                *iptr = x;

                if (hx < 0x7fff0000 || (hx == 0x7fff0000 && (xx.i[1] | xx.i[2] |
                    xx.i[3]) == 0)) {
                if (hx < 0x7fff0000 || (hx == 0x7fff0000 &&
                        (xx.i[1] | xx.i[2] | xx.i[3]) == 0)) {
                        xx.i[0] &= 0x80000000;
                        xx.i[1] = xx.i[2] = xx.i[3] = 0;
```



```
                }

#endif /* ! codereview */
                return (xx.q);
        }

        if (hx < 0x3fff0000) {          /* |x| < 1 */
                xx.i[0] &= 0x80000000;
                xx.i[1] = xx.i[2] = xx.i[3] = 0;
                *iptr = xx.q;
                return (x);
        }

        /* split x at the binary point */
        s = xx.i[0] & 0x80000000;

#endif /* ! codereview */
        if (hx < 0x40100000) {
                yy.i[0] = xx.i[0] & ~((1 << (0x400f - (hx >> 16))) - 1);
                yy.i[1] = yy.i[2] = yy.i[3] = 0;
        } else if (hx < 0x40300000) {
                yy.i[0] = xx.i[0];
                yy.i[1] = xx.i[1] & ~((1 << (0x402f - (hx >> 16))) - 1);
                yy.i[2] = yy.i[3] = 0;
        } else if (hx < 0x40500000) {
                yy.i[0] = xx.i[0];
                yy.i[1] = xx.i[1];
                yy.i[2] = xx.i[2] & ~((1 << (0x404f - (hx >> 16))) - 1);
                yy.i[3] = 0;
        } else {
                yy.i[0] = xx.i[0];
                yy.i[1] = xx.i[1];
                yy.i[2] = xx.i[2];
                yy.i[3] = xx.i[3] & ~((1 << (0x406f - (hx >> 16))) - 1);
        }

#endif /* ! codereview */
        *iptr = yy.q;

        /*
         * we could implement the following more efficiently than by using
         * software emulation of fsubq, but we'll do it this way for now
         * (and hope hardware support becomes commonplace)
         */
        xx.q -= yy.q;
        xx.i[0] = (xx.i[0] & ~0x80000000) | s;  /* keep sign of x */
        return (xx.q);
}
#elif defined(__x86)

long double
__modfl(long double x, long double *iptr)
{
__modfl(long double x, long double *iptr) {
        union {
                unsigned i[3];
                long double e;
        } xx, yy;

#endif /* ! codereview */
        unsigned hx, s;

        /*
         * It might be faster to use one of the x86 fpops instead of
         * the following.
```

```
 121          */
 122         xx.e = x;
 123         hx = xx.i[2] & 0x7fff;

 125         if (hx >= 0x403e) {             /* x is NaN, infinite, or integral */
 126                 *iptr = x;

 128                 if (hx < 0x7fff || (hx == 0x7fff && ((xx.i[1] << 1) |
 129                     xx.i[0]) == 0)) {
  46                 if (hx < 0x7fff || (hx == 0x7fff &&
  47                         ((xx.i[1] << 1) | xx.i[0]) == 0)) {
 130                         xx.i[2] &= 0x8000;
 131                         xx.i[1] = xx.i[0] = 0;
 132                 }

 134 #endif /* ! codereview */
 135                 return (xx.e);
 136         }

 138         if (hx < 0x3fff) {              /* |x| < 1 */
 139                 xx.i[2] &= 0x8000;
 140                 xx.i[1] = xx.i[0] = 0;
 141                 *iptr = xx.e;
 142                 return (x);
 143         }

 145         /* split x at the binary point */
 146         s = xx.i[2] & 0x8000;
 147         yy.i[2] = xx.i[2];

 149 #endif /* ! codereview */
 150         if (hx < 0x401f) {
 151                 yy.i[1] = xx.i[1] & ~((1 << (0x401e - hx)) - 1);
 152                 yy.i[0] = 0;
 153         } else {
 154                 yy.i[1] = xx.i[1];
 155                 yy.i[0] = xx.i[0] & ~((1 << (0x403e - hx)) - 1);
 156         }

 158 #endif /* ! codereview */
 159         *iptr = yy.e;
 160         xx.e -= yy.e;
 161         xx.i[2] = (xx.i[2] & ~0x8000) | s;      /* keep sign of x */
 162         return (xx.e);
 163 }

 164 #else
 165 #error Unknown architecture
 166 #endif
```

```
**********************************************************
    1497 Thu Jun  6 20:40:28 2019
new/usr/src/lib/libm/common/m9x/nan.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak nan = __nan

/*
 *  nan(c) returns a NaN.  This implementation ignores c.
 */

#include "libm.h"
#include <sys/isa_defs.h>

#if defined(__sparc)

static const union {
        unsigned i[2];
        double d;
} __nan_union = { 0x7fffffff, 0xffffffff };

#elif defined(__i386) || defined(__amd64)

static const union {
        unsigned i[2];
        double d;
} __nan_union = { 0xffffffff, 0x7fffffff };

#else
#error Unknown architecture
#endif

/* ARGSUSED0 */
double
__nan(const char *c)
```



```
{
__nan(const char *c) {
        return (__nan_union.d);
}
```

_____*unchanged_portion_omitted_*

```
**********************************************************
    1197 Thu Jun  6 20:40:28 2019
new/usr/src/lib/libm/common/m9x/nanf.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak nanf = __nanf

#include "libm.h"

static const union {
        unsigned i;
        float f;
} __nanf_union = { 0x7fffffff };

/* ARGSUSED0 */
float
__nanf(const char *c)
{
__nanf(const char *c) {
        return (__nanf_union.f);
}
______unchanged_portion_omitted_
```

```
**********************************************************
    1440 Thu Jun  6 20:40:29 2019
new/usr/src/lib/libm/common/m9x/nanl.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak nanl = __nanl

#include "libm.h"

#if defined(__sparc)

static const union {
        unsigned i[4];
        long double ld;
} __nanl_union = { 0x7fffffff, 0xffffffff, 0xffffffff, 0xffffffff };

#elif defined(__x86)

static const union {
        unsigned i[3];
        long double ld;
} __nanl_union = { 0xffffffff, 0xffffffff, 0x7fff };

#else
#error Unknown architecture
#endif

/* ARGSUSED0 */
long double
__nanl(const char *c)
{
__nanl(const char *c) {
        return (__nanl_union.ld);
}
______unchanged_portion_omitted_
```

```
**********************************************************
    3199 Thu Jun  6 20:40:29 2019
new/usr/src/lib/libm/common/m9x/nearbyint.c
11210 libm should be cstyle(10NBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak nearbyint = __nearbyint

/*
 * nearbyint(x) returns the nearest fp integer to x in the direction
 * corresponding to the current rounding direction without raising
 * the inexact exception.
 *
 * nearbyint(x) is x unchanged if x is +/-0 or +/-inf.  If x is NaN,
 * nearbyint(x) is also NaN.
 */

#include "libm.h"
#include <fenv.h>

double
__nearbyint(double x)
{
__nearbyint(double x) {
        union {
                unsigned i[2];
                double d;
        } xx;

#endif /* ! codereview */
        unsigned hx, sx, i, frac;
        int rm, j;

        xx.d = x;
        sx = xx.i[HIWORD] & 0x80000000;
        hx = xx.i[HIWORD] & ~0x80000000;
```



```
        /* handle trivial cases */
        if (hx >= 0x43300000) {         /* x is nan, inf, or already integral */
                if (hx >= 0x7ff00000)   /* x is inf or nan */
#if defined(FPADD_TRAPS_INCOMPLETE_ON_NAN)
                        return (hx >= 0x7ff80000 ? x : x + x);

#endif /* ! codereview */
                /* assumes sparc-like QNaN */
#else
                        return (x + x);
#endif
                return (x);
        } else if ((hx | xx.i[LOWORD]) == 0) {  /* x is zero */
        } else if ((hx | xx.i[LOWORD]) == 0)    /* x is zero */
                return (x);
        }
#endif /* ! codereview */

        /* get the rounding mode */
        rm = fegetround();

        /* flip the sense of directed roundings if x is negative */
        if (sx && (rm == FE_UPWARD || rm == FE_DOWNWARD))
                rm = (FE_UPWARD + FE_DOWNWARD) - rm;

        /* handle |x| < 1 */
        if (hx < 0x3ff00000) {
                if (rm == FE_UPWARD || (rm == FE_TONEAREST && (hx >=
                    0x3fe00000 && ((hx & 0xfffff) | xx.i[LOWORD]))))
                if (rm == FE_UPWARD || (rm == FE_TONEAREST &&
                        (hx >= 0x3fe00000 && ((hx & 0xfffff) | xx.i[LOWORD]))))
                        xx.i[HIWORD] = sx | 0x3ff00000;
                else
                        xx.i[HIWORD] = sx;

#endif /* ! codereview */
                xx.i[LOWORD] = 0;
                return (xx.d);
        }

        /* round x at the integer bit */
        j = 0x433 - (hx >> 20);

#endif /* ! codereview */
        if (j >= 32) {
                i = 1 << (j - 32);
                frac = ((xx.i[HIWORD] << 1) << (63 - j)) | (xx.i[LOWORD] >> (j -
                    32));

                frac = ((xx.i[HIWORD] << 1) << (63 - j)) |
                        (xx.i[LOWORD] >> (j - 32));
                if (xx.i[LOWORD] & (i - 1))
                        frac |= 1;

#endif /* ! codereview */
                if (!frac)
                        return (x);

#endif /* ! codereview */
                xx.i[LOWORD] = 0;
                xx.i[HIWORD] &= ~(i - 1);

                if ((rm == FE_UPWARD) || ((rm == FE_TONEAREST) && ((frac >
                    0x80000000u) || ((frac == 0x80000000) && (xx.i[HIWORD] &
                    i)))))
```

```
                if ((rm == FE_UPWARD) || ((rm == FE_TONEAREST) &&
                        ((frac > 0x80000000u) || ((frac == 0x80000000) &&
                        (xx.i[HIWORD] & i)))))
                        xx.i[HIWORD] += i;
        } else {
                i = 1 << j;
                frac = (xx.i[LOWORD] << 1) << (31 - j);

#endif /* ! codereview */
                if (!frac)
                        return (x);

#endif /* ! codereview */
                xx.i[LOWORD] &= ~(i - 1);

                if ((rm == FE_UPWARD) || ((rm == FE_TONEAREST) && (frac >
                    0x80000000u || ((frac == 0x80000000) && (xx.i[LOWORD] &
                    i))))) {
                if ((rm == FE_UPWARD) || ((rm == FE_TONEAREST) &&
                        (frac > 0x80000000u || ((frac == 0x80000000) &&
                        (xx.i[LOWORD] & i))))) {
                        xx.i[LOWORD] += i;

#endif /* ! codereview */
                        if (xx.i[LOWORD] == 0)
                                xx.i[HIWORD]++;
                }
        }

#endif /* ! codereview */
        return (xx.d);
}
```

```
**********************************************************
    3986 Thu Jun  6 20:40:30 2019
new/usr/src/lib/libm/common/m9x/nearbyintf.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak nearbyintf = __nearbyintf

#include "libm.h"
#include <fenv.h>

float
__nearbyintf(float x)
{
__nearbyintf(float x) {
        union {
                unsigned i;
                float f;
        } xx;

#endif /* ! codereview */
        unsigned hx, sx, i, frac;
        int rm;

        xx.f = x;
        sx = xx.i & 0x80000000;
        hx = xx.i & ~0x80000000;

        /* handle trivial cases */
        if (hx >= 0x4b000000) {         /* x is nan, inf, or already integral */
                if (hx > 0x7f800000)    /* x is nan */
                        return (x * x);         /* + -> * for Cheetah */

#endif /* ! codereview */
                return (x);
        } else if (hx == 0) {                   /* x is zero */
```



```
        } else if (hx == 0)             /* x is zero */
                return (x);
        }
#endif /* ! codereview */

        /* get the rounding mode */
        rm = fegetround();

        /* flip the sense of directed roundings if x is negative */
        if (sx && (rm == FE_UPWARD || rm == FE_DOWNWARD))
                rm = (FE_UPWARD + FE_DOWNWARD) - rm;

        /* handle |x| < 1 */
        if (hx < 0x3f800000) {
                if (rm == FE_UPWARD || (rm == FE_TONEAREST && hx > 0x3f000000))
                        xx.i = sx | 0x3f800000;
                else
                        xx.i = sx;

#endif /* ! codereview */
                return (xx.f);
        }

        /* round x at the integer bit */
        i = 1 << (0x96 - (hx >> 23));
        frac = hx & (i - 1);

#endif /* ! codereview */
        if (!frac)
                return (x);

        hx &= ~(i - 1);

#endif /* ! codereview */
        if (rm == FE_UPWARD || (rm == FE_TONEAREST && (frac > (i >> 1) ||
            ((frac == (i >> 1)) && (hx & i)))))
                xx.i = sx | (hx + i);
        else
                xx.i = sx | hx;

#endif /* ! codereview */
        return (xx.f);
}

#if 0

/*
 * Alternate implementations for SPARC, x86, using fp ops.  These may
 * be faster depending on how expensive saving and restoring the fp
 * modes and status flags is.
 */

#include "libm.h"
#include "fma.h"

#if defined(__sparc)

float
__nearbyintf(float x)
{
__nearbyintf(float x) {
        union {
                unsigned i;
                float f;
        } xx, yy;
```

```
#endif /* ! codereview */
        float z;
        unsigned hx, sx, fsr, oldfsr;
        int rm;

        xx.f = x;
        sx = xx.i & 0x80000000;
        hx = xx.i & ~0x80000000;

        /* handle trivial cases */
        if (hx >= 0x4b000000)           /* x is nan, inf, or already integral */
                return (x + 0.0f);
        else if (hx == 0)               /* x is zero */
                return (x);

        /* save the fsr */
        __fenv_getfsr(&oldfsr);

        /* handle |x| < 1 */
        if (hx < 0x3f800000) {
                /* flip the sense of directed roundings if x is negative */
                rm = oldfsr >> 30;

#endif /* ! codereview */
                if (sx)
                        rm ^= rm >> 1;

#endif /* ! codereview */
                if (rm == FSR_RP || (rm == FSR_RN && hx > 0x3f000000))
                        xx.i = sx | 0x3f800000;
                else
                        xx.i = sx;

#endif /* ! codereview */
                return (xx.f);
        }

        /* clear the inexact trap */
        fsr = oldfsr & ~FSR_NXM;
        __fenv_setfsr(&fsr);

        /* round x at the integer bit */
        yy.i = sx | 0x4b000000;
        z = (x + yy.f) - yy.f;

        /* restore the old fsr */
        __fenv_setfsr(&oldfsr);

        return (z);
}
#elif defined(__x86)

/* inline template */
extern long double frndint(long double);

float
__nearbyintf(float x)
{
__nearbyintf(float x) {
        long double z;
        unsigned oldcwsw, cwsw;

        /* save the control and status words, mask the inexact exception */
        __fenv_getcwsw(&oldcwsw);
        cwsw = oldcwsw | 0x00200000;
```



```
        __fenv_setcwsw(&cwsw);

        z = frndint((long double)x);
        z = frndint((long double) x);

        /*
         * restore the control and status words, preserving all but the
         * inexact flag
         */
        __fenv_getcwsw(&cwsw);
        oldcwsw |= (cwsw & 0x1f);
        __fenv_setcwsw(&oldcwsw);

        /* note: the value of z is representable in single precision */
        return (z);
}
#else
#error Unknown architecture
#endif

#endif
```

```
**********************************************************
    4449 Thu Jun  6 20:40:30 2019
new/usr/src/lib/libm/common/m9x/nearbyintl.c
11210 libm should be cstyle(10NBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak nearbyintl = __nearbyintl

#include "libm.h"
#include "fma.h"
#include "fenv_inlines.h"

#if defined(__sparc)

static union {
        unsigned i;
        float f;
} snan = { 0x7f800001 };

long double
__nearbyintl(long double x)
{
__nearbyintl(long double x) {
        union {
                unsigned i[4];
                long double q;
        } xx;

#endif /* ! codereview */
        unsigned hx, sx, i, frac;
        unsigned int fsr;
        int rm, j;
        volatile float dummy;

        xx.q = x;
        sx = xx.i[0] & 0x80000000;
```



```
        hx = xx.i[0] & ~0x80000000;

        /* handle trivial cases */
        if (hx >= 0x406f0000) {         /* x is nan, inf, or already integral */
                /* check for signaling nan */
                if ((hx > 0x7fff0000 || (hx == 0x7fff0000 && (xx.i[1] |
                    xx.i[2] | xx.i[3]))) && !(hx & 0x8000)) {
                if ((hx > 0x7fff0000 || (hx == 0x7fff0000 &&
                        (xx.i[1] | xx.i[2] | xx.i[3]))) && !(hx & 0x8000)) {
                        dummy = snan.f;
                        dummy += snan.f;
                        xx.i[0] = sx | hx | 0x8000;
                }

#endif /* ! codereview */
                return (xx.q);
        } else if ((hx | xx.i[1] | xx.i[2] | xx.i[3]) == 0) {   /* x is zero */
        } else if ((hx | xx.i[1] | xx.i[2] | xx.i[3]) == 0)     /* x is zero */
                return (x);
        }
#endif /* ! codereview */

        /* get the rounding mode */
        __fenv_getfsr32(&fsr);
        rm = fsr >> 30;

        /* flip the sense of directed roundings if x is negative */
        if (sx)
                rm ^= rm >> 1;

        /* handle |x| < 1 */
        if (hx < 0x3fff0000) {
                if (rm == FSR_RP || (rm == FSR_RN && (hx >= 0x3ffe0000 && ((hx &
                    0xffff) | xx.i[1] | xx.i[2] | xx.i[3]))))
                if (rm == FSR_RP || (rm == FSR_RN && (hx >= 0x3ffe0000 &&
                        ((hx & 0xffff) | xx.i[1] | xx.i[2] | xx.i[3]))))
                        xx.i[0] = sx | 0x3fff0000;
                else
                        xx.i[0] = sx;

#endif /* ! codereview */
                xx.i[1] = xx.i[2] = xx.i[3] = 0;
                return (xx.q);
        }

        /* round x at the integer bit */
        j = 0x406f - (hx >> 16);

#endif /* ! codereview */
        if (j >= 96) {
                i = 1 << (j - 96);
                frac = ((xx.i[0] << 1) << (127 - j)) | (xx.i[1] >> (j - 96));

#endif /* ! codereview */
                if ((xx.i[1] & (i - 1)) | xx.i[2] | xx.i[3])
                        frac |= 1;

#endif /* ! codereview */
                if (!frac)
                        return (x);

#endif /* ! codereview */
                xx.i[1] = xx.i[2] = xx.i[3] = 0;
                xx.i[0] &= ~(i - 1);

#endif /* ! codereview */
```

```
                if (rm == FSR_RP || (rm == FSR_RN && (frac > 0x80000000u ||
                    (frac == 0x80000000 && (xx.i[0] & i)))))
                        xx.i[0] += i;
        } else if (j >= 64) {
                i = 1 << (j - 64);
                frac = ((xx.i[1] << 1) << (95 - j)) | (xx.i[2] >> (j - 64));

#endif /* ! codereview */
                if ((xx.i[2] & (i - 1)) | xx.i[3])
                        frac |= 1;

#endif /* ! codereview */
                if (!frac)
                        return (x);

#endif /* ! codereview */
                xx.i[2] = xx.i[3] = 0;
                xx.i[1] &= ~(i - 1);

#endif /* ! codereview */
                if (rm == FSR_RP || (rm == FSR_RN && (frac > 0x80000000u ||
                    (frac == 0x80000000 && (xx.i[1] & i))))) {
                        xx.i[1] += i;

#endif /* ! codereview */
                        if (xx.i[1] == 0)
                                xx.i[0]++;
                }
        } else if (j >= 32) {
                i = 1 << (j - 32);
                frac = ((xx.i[2] << 1) << (63 - j)) | (xx.i[3] >> (j - 32));

#endif /* ! codereview */
                if (xx.i[3] & (i - 1))
                        frac |= 1;

#endif /* ! codereview */
                if (!frac)
                        return (x);

#endif /* ! codereview */
                xx.i[3] = 0;
                xx.i[2] &= ~(i - 1);

#endif /* ! codereview */
                if (rm == FSR_RP || (rm == FSR_RN && (frac > 0x80000000u ||
                    (frac == 0x80000000 && (xx.i[2] & i))))) {
                        xx.i[2] += i;

#endif /* ! codereview */
                        if (xx.i[2] == 0)
                                if (++xx.i[1] == 0)
                                        xx.i[0]++;
                }
        } else {
                i = 1 << j;
                frac = (xx.i[3] << 1) << (31 - j);

#endif /* ! codereview */
                if (!frac)
                        return (x);

#endif /* ! codereview */
                xx.i[3] &= ~(i - 1);

#endif /* ! codereview */
```



```
                if (rm == FSR_RP || (rm == FSR_RN && (frac > 0x80000000u ||
                    (frac == 0x80000000 && (xx.i[3] & i))))) {
                        xx.i[3] += i;

#endif /* ! codereview */
                        if (xx.i[3] == 0)
                                if (++xx.i[2] == 0)
                                        if (++xx.i[1] == 0)
                                                xx.i[0]++;
                }
        }

        return (xx.q);
}

#elif defined(__x86)

/* inline template */
extern long double frndint(long double);

long double
__nearbyintl(long double x)
{
__nearbyintl(long double x) {
        long double z;
        unsigned oldcwsw, cwsw;

        /* save the control and status words, mask the inexact exception */
        __fenv_getcwsw(&oldcwsw);
        cwsw = oldcwsw | 0x00200000;
        __fenv_setcwsw(&cwsw);

        z = frndint(x);

        /*
         * restore the control and status words, preserving all but the
         * inexact flag
         */
        __fenv_getcwsw(&cwsw);
        oldcwsw |= (cwsw & 0x1f);
        __fenv_setcwsw(&oldcwsw);

        return (z);
}

#else
#error Unknown architecture
#endif
```

```
**********************************************************
    4655 Thu Jun  6 20:40:31 2019
new/usr/src/lib/libm/common/m9x/nexttoward.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak nexttoward = __nexttoward

/*
 * nexttoward(x, y) delivers the next representable number after x
 * in the direction of y.  If x and y are both zero, the result is
 * zero with the same sign as y.  If either x or y is NaN, the result
 * is NaN.
 *
 * If x != y and the result is infinite, overflow is raised; if
 * x != y and the result is subnormal or zero, underflow is raised.
 * (This is wrong, but it's what C99 apparently wants.)
 */

#include "libm.h"

#if defined(__sparc)

static union {
        unsigned i[2];
        double d;
} C[] = {
        0x00100000, 0, 0x7fe00000, 0, 0x7fffffff, 0xffffffff
        0x00100000, 0,
        0x7fe00000, 0,
        0x7fffffff, 0xffffffff
};

#define tiny            C[0].d
#define huge            C[1].d
#define qnan            C[2].d
```



```
enum fcc_type {
        fcc_equal = 0, fcc_less = 1, fcc_greater = 2, fcc_unordered = 3
        fcc_equal = 0,
        fcc_less = 1,
        fcc_greater = 2,
        fcc_unordered = 3
};

#ifdef __sparcv9
#define _Q_cmp          _Qp_cmp
#endif

extern enum fcc_type _Q_cmp(const long double *, const long double *);

double
__nexttoward(double x, long double y)
{
__nexttoward(double x, long double y) {
        union {
                unsigned i[2];
                double d;
        } xx;
        union {
                unsigned i[4];
                long double q;
        } yy;

#endif /* ! codereview */
        long double lx;
        unsigned hx;
        volatile double dummy;
        enum fcc_type rel;

        /*
         * It would be somewhat more efficient to check for NaN and
         * zero operands before converting x to long double and then
         * to code the comparison in line rather than calling _Q_cmp.
         * However, since this code probably won't get used much,
         * I'm opting in favor of simplicity instead.
         */
        lx = xx.d = x;
        hx = (xx.i[0] & ~0x80000000) | xx.i[1];

        /* check for each of four possible orderings */
        rel = _Q_cmp(&lx, &y);

#endif /* ! codereview */
        if (rel == fcc_unordered)
                return (qnan);

        if (rel == fcc_equal) {
                if (hx == 0) {  /* x is zero; return zero with y's sign */
                        yy.q = y;
                        xx.i[0] = yy.i[0];
                        return (xx.d);
                }

#endif /* ! codereview */
                return (x);
        }

        if (rel == fcc_less) {
                if (hx == 0) {                  /* x is zero */
                        xx.i[0] = 0;
                        xx.i[1] = 0x00000001;
```

```
                } else if ((int)xx.i[0] >= 0) { /* x is positive */
                        if (++xx.i[1] == 0)
                                xx.i[0]++;
                } else {
                        if (xx.i[1]-- == 0)
                                xx.i[0]--;
                }
        } else {
                if (hx == 0) {                  /* x is zero */
                        xx.i[0] = 0x80000000;
                        xx.i[1] = 0x00000001;
                } else if ((int)xx.i[0] >= 0) { /* x is positive */
                        if (xx.i[1]-- == 0)
                                xx.i[0]--;
                } else {
                        if (++xx.i[1] == 0)
                                xx.i[0]++;
                }
        }

        /* raise exceptions as needed */
        hx = xx.i[0] & ~0x80000000;

#endif /* ! codereview */
        if (hx == 0x7ff00000) {
                dummy = huge;
                dummy *= huge;
        } else if (hx < 0x00100000) {
                dummy = tiny;
                dummy *= tiny;
        }

        return (xx.d);
}
#elif defined(__x86)

static union {
        unsigned i[2];
        double d;
} C[] = {
        0, 0x00100000, 0, 0x7fe00000, };
        0, 0x00100000,
        0, 0x7fe00000,
};

#define tiny            C[0].d
#define huge            C[1].d

double
__nexttoward(double x, long double y)
{
__nexttoward(double x, long double y) {
        union {
                unsigned i[2];
                double d;
        } xx;

#endif /* ! codereview */
        unsigned hx;
        long double lx;
        volatile double dummy;

        lx = xx.d = x;
        hx = (xx.i[1] & ~0x80000000) | xx.i[0];
```



```
        /* check for each of four possible orderings */
        if (isunordered(lx, y))
                return ((double)(lx + y));
                return ((double) (lx + y));

        if (lx == y)
                return ((double)y);
                return ((double) y);

        if (lx < y) {
                if (hx == 0) {                  /* x is zero */
                        xx.i[0] = 0x00000001;
                        xx.i[1] = 0;
                } else if ((int)xx.i[1] >= 0) { /* x is positive */
                        if (++xx.i[0] == 0)
                                xx.i[1]++;
                } else {
                        if (xx.i[0]-- == 0)
                                xx.i[1]--;
                }
        } else {
                if (hx == 0) {                  /* x is zero */
                        xx.i[0] = 0x00000001;
                        xx.i[1] = 0x80000000;
                } else if ((int)xx.i[1] >= 0) { /* x is positive */
                        if (xx.i[0]-- == 0)
                                xx.i[1]--;
                } else {
                        if (++xx.i[0] == 0)
                                xx.i[1]++;
                }
        }

        /* raise exceptions as needed */
        hx = xx.i[1] & ~0x80000000;

#endif /* ! codereview */
        if (hx == 0x7ff00000) {
                dummy = huge;
                dummy *= huge;
        } else if (hx < 0x00100000) {
                dummy = tiny;
                dummy *= tiny;
        }

        return (xx.d);
}
#else
#error Unknown architecture
#endif
```

```
**********************************************************
    3659 Thu Jun  6 20:40:32 2019
new/usr/src/lib/libm/common/m9x/nexttowardf.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak nexttowardf = __nexttowardf

#include "libm.h"

static union {
        unsigned i;
        float f;
} C[] = {
        0x00800000, 0x7f000000, 0x7fffffff
        0x00800000,
        0x7f000000,
        0x7fffffff
};

#define tiny            C[0].f
#define huge            C[1].f
#define qnan            C[2].f

#if defined(__sparc)

enum fcc_type {
        fcc_equal = 0, fcc_less = 1, fcc_greater = 2, fcc_unordered = 3
        fcc_equal = 0,
        fcc_less = 1,
        fcc_greater = 2,
        fcc_unordered = 3
};

#ifdef __sparcv9
#define _Q_cmp  _Qp_cmp
```



```
#endif

extern enum fcc_type _Q_cmp(const long double *, const long double *);

float
__nexttowardf(float x, long double y)
{
__nexttowardf(float x, long double y) {
        union {
                unsigned i;
                float f;
        } xx;
        union {
                unsigned i[4];
                long double q;
        } yy;

#endif /* ! codereview */
        long double lx;
        unsigned hx;
        volatile float dummy;
        enum fcc_type rel;

        /*
         * It would be somewhat more efficient to check for NaN and
         * zero operands before converting x to long double and then
         * to code the comparison in line rather than calling _Q_cmp.
         * However, since this code probably won't get used much,
         * I'm opting in favor of simplicity instead.
         */
        lx = xx.f = x;
        hx = xx.i & ~0x80000000;

        /* check for each of four possible orderings */
        rel = _Q_cmp(&lx, &y);

#endif /* ! codereview */
        if (rel == fcc_unordered)
                return (qnan);

        if (rel == fcc_equal) {
                if (hx == 0) {  /* x is zero; return zero with y's sign */
                        yy.q = y;
                        xx.i = yy.i[0];
                        return (xx.f);
                }

#endif /* ! codereview */
                return (x);
        }

        if (rel == fcc_less) {
                if (hx == 0)                    /* x is zero */
                        xx.i = 0x00000001;
                else if ((int)xx.i >= 0)        /* x is positive */
                else if ((int) xx.i >= 0)       /* x is positive */
                        xx.i++;
                else
                        xx.i--;
        } else {
                if (hx == 0)                    /* x is zero */
                        xx.i = 0x80000001;
                else if ((int)xx.i >= 0)        /* x is positive */
                else if ((int) xx.i >= 0)       /* x is positive */
                        xx.i--;
                else
```

```
                        xx.i++;
        }

        /* raise exceptions as needed */
        hx = xx.i & ~0x80000000;

#endif /* ! codereview */
        if (hx == 0x7f800000) {
                dummy = huge;
                dummy *= huge;
        } else if (hx < 0x00800000) {
                dummy = tiny;
                dummy *= tiny;
        }

        return (xx.f);
}
#elif defined(__x86)

float
__nexttowardf(float x, long double y)
{
__nexttowardf(float x, long double y) {
        union {
                unsigned i;
                float f;
        } xx;

#endif /* ! codereview */
        unsigned hx;
        long double lx;
        volatile float dummy;

        lx = xx.f = x;
        hx = xx.i & ~0x80000000;

        /* check for each of four possible orderings */
        if (isunordered(lx, y))
                return ((float)(lx + y));
                return ((float) (lx + y));

        if (lx == y)
                return ((float)y);
                return ((float) y);

        if (lx < y) {
                if (hx == 0)                    /* x is zero */
                        xx.i = 0x00000001;
                else if ((int)xx.i >= 0)        /* x is positive */
                else if ((int) xx.i >= 0)       /* x is positive */
                        xx.i++;
                else
                        xx.i--;
        } else {
                if (hx == 0)                    /* x is zero */
                        xx.i = 0x80000001;
                else if ((int)xx.i >= 0)        /* x is positive */
                else if ((int) xx.i >= 0)       /* x is positive */
                        xx.i--;
                else
                        xx.i++;
        }

        /* raise exceptions as needed */
        hx = xx.i & ~0x80000000;
```



```
#endif /* ! codereview */
        if (hx == 0x7f800000) {
                dummy = huge;
                dummy *= huge;
        } else if (hx < 0x00800000) {
                dummy = tiny;
                dummy *= tiny;
        }

        return (xx.f);
}
#else
#error Unknown architecture
#endif
```

```
**********************************************************
    2740 Thu Jun  6 20:40:32 2019
new/usr/src/lib/libm/common/m9x/nexttowardl.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __nexttowardl = nexttowardl

#include "libm.h"
#include <float.h>              /* LDBL_MAX, LDBL_MIN */

#if defined(__sparc)
#define n0      0
#define n1      1
#define n2      2
#define n3      3
#define X86PDNRM1(x)
#define INC(px) { \
                        if (++px[n3] == 0) \
                                if (++px[n2] == 0) \
                                        if (++px[n1] == 0) \
                                                ++px[n0]; \
                }
#define DEC(px) { \
                        if (--px[n3] == 0xffffffff) \
                                if (--px[n2] == 0xffffffff) \
                                        if (--px[n1] == 0xffffffff) \
                                                --px[n0]; \
                }
#elif defined(__x86)
#define n0      2
#define n1      1
#define n2      0
#define n3      0
/*
 * if pseudo-denormal, replace by the equivalent normal
 */
#define X86PDNRM1(x)    if (XBIASED_EXP(x) == 0 && (((int *)&x)[1] & \
```



```
#define X86PDNRM1(x)    if (XBIASED_EXP(x) == 0 && (((int *) &x)[1] & \
                                0x80000000) != 0) \
                                ((int *)&x)[2] |= 1
                                ((int *) &x)[2] |= 1
#define INC(px) { \
                        if (++px[n2] == 0) \
                                if ((++px[n1] & ~0x80000000) == 0) \
                                        px[n1] = 0x80000000, ++px[n0]; \
                }
#define DEC(px) { \
                        if (--px[n2] == 0xffffffff) \
                                if (--px[n1] == 0x7fffffff) \
                                        if ((--px[n0] & 0x7fff) != 0) \
                                                px[n1] |= 0x80000000; \
                }
#endif

long double
nexttowardl(long double x, long double y)
{
        int *px = (int *)&x;
        int *py = (int *)&y;
nexttowardl(long double x, long double y) {
        int *px = (int *) &x;
        int *py = (int *) &y;

        if (x == y)
                return (y);             /* C99 requirement */
        if (x != x || y != y)
                return (x * y);

        if (ISZEROL(x)) {       /* x == 0.0 */
                px[n0] = py[n0] & XSGNMSK;
                px[n1] = px[n2] = 0;
                px[n3] = 1;
        } else {
                X86PDNRM1(x);
                if ((px[n0] & XSGNMSK) == 0) {  /* x > 0.0 */
                        if (x > y)      /* x > y */
                                DEC(px)
                        else
                                INC(px)
                } else {
                        if (x < y)      /* x < y */
                                DEC(px)
                        else
                                INC(px)
                }
        }
#ifndef lint
        {
                volatile long double dummy __unused;
                int k = XBIASED_EXP(x);

                if (k == 0)
                        dummy = LDBL_MIN * copysignl(LDBL_MIN, x);
                else if (k == 0x7fff)
                        dummy = LDBL_MAX * copysignl(LDBL_MAX, x);
        }
#endif
        return (x);
}
______unchanged_portion_omitted_
```

```
**********************************************************
    5995 Thu Jun  6 20:40:33 2019
new/usr/src/lib/libm/common/m9x/remquo.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __remquo = remquo


/* INDENT OFF */
/*
 * double remquo(double x, double y, int *quo) return remainder(x,y) and an
 * integer pointer quo such that *quo = N mod {2**31}, where N is the
 * exact integral part of x/y rounded to nearest even.
 *
 * remquo call internal fmodquo
 */
/* INDENT ON */

#include "libm.h"
#include "libm_protos.h"
#include <math.h>                       /* fabs() */
#include <sys/isa_defs.h>

#if defined(_BIG_ENDIAN)
#define HIWORD          0
#define LOWORD          1
#else
#define HIWORD          1
#define LOWORD          0
#endif
#define __HI(x)         ((int *)&x)[HIWORD]
#define __LO(x)         ((int *)&x)[LOWORD]
#define __HI(x) ((int *) &x)[HIWORD]
#define __LO(x) ((int *) &x)[LOWORD]
```



```
static const double one = 1.0, Zero[] = {0.0, -0.0};

static const double one = 1.0, Zero[] = { 0.0, -0.0 };
#endif /* ! codereview */
static double
fmodquo(double x, double y, int *quo)
{
fmodquo(double x, double y, int *quo) {
        int n, hx, hy, hz, ix, iy, sx, sq, i, m;
        unsigned lx, ly, lz;

        hx = __HI(x);                   /* high word of x */
        lx = __LO(x);                   /* low  word of x */
        hy = __HI(y);                   /* high word of y */
        ly = __LO(y);                   /* low  word of y */
        sx = hx & 0x80000000;           /* sign of x */
        sq = (hx ^ hy) & 0x80000000;    /* sign of x/y */
        hx ^= sx;                       /* |x| */
        hy &= 0x7fffffff;               /* |y| */

        /* purge off exception values */
        *quo = 0;

#endif /* ! codereview */
        if ((hy | ly) == 0 || hx >= 0x7ff00000 ||       /* y=0, or x !finite */
            (hy | ((ly | -ly) >> 31)) > 0x7ff00000)     /* or y is NaN */
                return ((x * y) / (x * y));

#endif /* ! codereview */
        if (hx <= hy) {
                if (hx < hy || lx < ly)
                        return (x);     /* |x|<|y| return x */

#endif /* ! codereview */
                if (lx == ly) {
                        *quo = 1 + (sq >> 30);
                        /* |x|=|y| return x*0 */
                        return (Zero[(unsigned)sx >> 31]);
                        return (Zero[(unsigned) sx >> 31]);
                }
        }

        /* determine ix = ilogb(x) */
        if (hx < 0x00100000) {          /* subnormal x */
                if (hx == 0) {
                        for (ix = -1043, i = lx; i > 0; i <<= 1)
                                ix -= 1;
                } else {
                        for (ix = -1022, i = (hx << 11); i > 0; i <<= 1)
                                ix -= 1;
                }
        } else {
        } else
                ix = (hx >> 20) - 1023;
        }
#endif /* ! codereview */

        /* determine iy = ilogb(y) */
        if (hy < 0x00100000) {          /* subnormal y */
                if (hy == 0) {
                        for (iy = -1043, i = ly; i > 0; i <<= 1)
                                iy -= 1;
                } else {
                        for (iy = -1022, i = (hy << 11); i > 0; i <<= 1)
                                iy -= 1;
                }
```

```
119         } else {
 82         } else
120                 iy = (hy >> 20) - 1023;
121         }
122 #endif /* ! codereview */

124         /* set up {hx,lx}, {hy,ly} and align y to x */
125         if (ix >= -1022) {
 84         if (ix >= -1022)
126                 hx = 0x00100000 | (0x000fffff & hx);
127         } else {                        /* subnormal x, shift x to normal */
 86         else {                  /* subnormal x, shift x to normal */
128                 n = -1022 - ix;

130 #endif /* ! codereview */
131                 if (n <= 31) {
132                         hx = (hx << n) | (lx >> (32 - n));
133                         lx <<= n;
134                 } else {
135                         hx = lx << (n - 32);
136                         lx = 0;
137                 }
138         }

140         if (iy >= -1022) {
 88         if (iy >= -1022)
141                 hy = 0x00100000 | (0x000fffff & hy);
142         } else {                        /* subnormal y, shift y to normal */
 90         else {                  /* subnormal y, shift y to normal */
143                 n = -1022 - iy;

145 #endif /* ! codereview */
146                 if (n <= 31) {
147                         hy = (hy << n) | (ly >> (32 - n));
148                         ly <<= n;
149                 } else {
150                         hy = ly << (n - 32);
151                         ly = 0;
152                 }
153         }

155         /* fix point fmod */
156         n = ix - iy;
157         m = 0;

159 #endif /* ! codereview */
160         while (n--) {
161                 hz = hx - hy;
162                 lz = lx - ly;

164 #endif /* ! codereview */
165                 if (lx < ly)
166                         hz -= 1;

168 #endif /* ! codereview */
169                 if (hz < 0) {
170                         hx = hx + hx + (lx >> 31);
171                         lx = lx + lx;
172                 } else {
173                         m += 1;

175 #endif /* ! codereview */
176                         if ((hz | lz) == 0) {   /* return sign(x)*0 */
177                                 if (n < 31)
178                                         m <<= 1 + n;
179                                 else
```



```
180                                         m = 0;

182 #endif /* ! codereview */
183                                 m &= 0x7fffffff;
184                                 *quo = sq >= 0 ? m : -m;
185                                 return (Zero[(unsigned)sx >> 31]);
 92                                 return (Zero[(unsigned) sx >> 31]);
186                         }

188 #endif /* ! codereview */
189                         hx = hz + hz + (lz >> 31);
190                         lx = lz + lz;
191                 }

193 #endif /* ! codereview */
194                 m += m;
195         }

197 #endif /* ! codereview */
198         hz = hx - hy;
199         lz = lx - ly;

201 #endif /* ! codereview */
202         if (lx < ly)
203                 hz -= 1;

205 #endif /* ! codereview */
206         if (hz >= 0) {
207                 hx = hz;
208                 lx = lz;
209                 m += 1;
210         }

212 #endif /* ! codereview */
213         m &= 0x7fffffff;
214         *quo = sq >= 0 ? m : -m;

216         /* convert back to floating value and restore the sign */
217         if ((hx | lx) == 0)             /* return sign(x)*0 */
218                 return (Zero[(unsigned)sx >> 31]);

 94         if ((hx | lx) == 0) {   /* return sign(x)*0 */
 95                 return (Zero[(unsigned) sx >> 31]);
 96         }
220         while (hx < 0x00100000) {       /* normalize x */
221                 hx = hx + hx + (lx >> 31);
222                 lx = lx + lx;
223                 iy -= 1;
224         }

226 #endif /* ! codereview */
227         if (iy >= -1022) {              /* normalize output */
228                 hx = (hx - 0x00100000) | ((iy + 1023) << 20);
229                 __HI(x) = hx | sx;
230                 __LO(x) = lx;
231         } else {                        /* subnormal output */
232                 n = -1022 - iy;

234 #endif /* ! codereview */
235                 if (n <= 20) {
236                         lx = (lx >> n) | ((unsigned)hx << (32 - n));
102                         lx = (lx >> n) | ((unsigned) hx << (32 - n));
237                         hx >>= n;
238                 } else if (n <= 31) {
239                         lx = (hx << (32 - n)) | (lx >> n);
240                         hx = sx;
```

```
                } else {
                        lx = hx >> (n - 32);
                        hx = sx;
                }
```

```
#endif /* ! codereview */
                __HI(x) = hx | sx;
                __LO(x) = lx;
                x *= one;               /* create necessary signal */
        }

#endif /* ! codereview */
        return (x);                     /* exact output */
}

#define zero    Zero[0]

double
remquo(double x, double y, int *quo)
{
remquo(double x, double y, int *quo) {
        int hx, hy, sx, sq;
        double v;
        unsigned ly;

        hx = __HI(x);                   /* high word of x */
        hy = __HI(y);                   /* high word of y */
        ly = __LO(y);                   /* low  word of y */
        sx = hx & 0x80000000;           /* sign of x */
        sq = (hx ^ hy) & 0x80000000;    /* sign of x/y */
        hx ^= sx;                       /* |x| */
        hy &= 0x7fffffff;               /* |y| */

        /* purge off exception values */
        *quo = 0;
```

```
#endif /* ! codereview */
        if ((hy | ly) == 0 || hx >= 0x7ff00000 ||       /* y=0, or x !finite */
            (hy | ((ly | -ly) >> 31)) > 0x7ff00000)     /* or y is NaN */
                return ((x * y) / (x * y));

        y = fabs(y);
        x = fabs(x);

#endif /* ! codereview */
        if (hy <= 0x7fdfffff) {
                x = fmodquo(x, y + y, quo);
                *quo = ((*quo) & 0x3fffffff) << 1;
        }

#endif /* ! codereview */
        if (hy < 0x00200000) {
                if (x + x > y) {
                        *quo += 1;

#endif /* ! codereview */
                        if (x == y)
                                x = zero;
                        else
                                x -= y;

#endif /* ! codereview */
                        if (x + x >= y) {
                                x -= y;
                                *quo += 1;
                        }
```



```
                }
        } else {
                v = 0.5 * y;

#endif /* ! codereview */
                if (x > v) {
                        *quo += 1;

#endif /* ! codereview */
                        if (x == y)
                                x = zero;
                        else
                                x -= y;

#endif /* ! codereview */
                        if (x >= v) {
                                x -= y;
                                *quo += 1;
                        }
                }
        }

#endif /* ! codereview */
        if (sq != 0)
                *quo = -(*quo);

#endif /* ! codereview */
        return (sx == 0 ? x : -x);
}
```

```
**********************************************************
    5165 Thu Jun  6 20:40:33 2019
new/usr/src/lib/libm/common/m9x/remquof.c
11210 libm should be cstyle(10NBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __remquof = remquof


/* INDENT OFF */
/*
 * float remquof(float x, float y, int *quo) return remainderf(x,y) and an
 * integer pointer quo such that *quo = N mod (2**31),  where N is the
 * exact integeral part of x/y rounded to nearest even.
 *
 * remquof call internal fmodquof
 */

#include "libm.h"
#include "libm_protos.h"
#include <math.h>

#endif /* ! codereview */
extern float fabsf(float);
static const int is = (int)0x80000000,

static const int
        is = (int) 0x80000000,
        im = 0x007fffff,
        ii = 0x7f800000,
        iu = 0x00800000;

static const float zero = 0.0F, half = 0.5F;
/* INDENT ON */

static float
```



```
fmodquof(float x, float y, int *quo)
{
fmodquof(float x, float y, int *quo) {
        float w;
        int hx, ix, iy, iz, k, ny, nd, m, sq;

        hx = *(int *)&x;
        hx = *(int *) &x;
        ix = hx & 0x7fffffff;
        iy = *(int *)&y;
        iy = *(int *) &y;
        sq = (iy ^ hx) & is;            /* sign of x/y */
        iy &= 0x7fffffff;

        /* purge off exception values */
        *quo = 0;

#endif /* ! codereview */
        if (ix >= ii || iy > ii || iy == 0) {
                w = x * y;
                w = w / w;
        } else if (ix <= iy) {
                if (ix < iy) {
                if (ix < iy)
                        w = x;          /* return x if |x|<|y| */
                } else {
                else {
                        *quo = 1 + (sq >> 30);
                        w = zero * x;   /* return sign(x)*0.0  */
                }
        } else {

                /* INDENT OFF */
                /*
                 * scale x,y to "normal" with
                 *      ny = exponent of y
                 *      nd = exponent of x minus exponent of y
                 */
                /* INDENT ON */
                ny = iy >> 23;
                k = ix >> 23;

                /* special case for subnormal y or x */
                if (ny == 0) {
                        ny = 1;

#endif /* ! codereview */
                        while (iy < iu) {
                                ny -= 1;
                                iy += iy;
                        }

#endif /* ! codereview */
                        nd = k - ny;

#endif /* ! codereview */
                        if (k == 0) {
                                nd += 1;

#endif /* ! codereview */
                                while (ix < iu) {
                                        nd -= 1;
                                        ix += ix;
                                }
                        } else {
                        } else
```

```
                                ix = iu | (ix & im);
                        }
#endif /* ! codereview */
                } else {
                        nd = k - ny;
                        ix = iu | (ix & im);
                        iy = iu | (iy & im);
                }

                /*
                 * fix point fmod for normalized ix and iy
                 */

                /* INDENT OFF */
                /* fix point fmod for normalized ix and iy */
                /*
                 * while (nd--) {
                 *      iz = ix - iy;
                 *      if (iz < 0)
                 *              ix = ix + ix;
                 *      else if (iz == 0) {
                 *              *(int *) &w = is & hx;
                 *              return w;
                 *      } else
                 *              ix = iz + iz;
                 * }
                 */

                /*
                 * unroll the above loop 4 times to gain performance
                 */
                /* INDENT ON */
                /* unroll the above loop 4 times to gain performance */
                m = 0;
                k = nd >> 2;
                nd -= (k << 2);

#endif /* ! codereview */
                while (k--) {
                        iz = ix - iy;

#endif /* ! codereview */
                        if (iz >= 0) {
                                m += 1;
                                ix = iz + iz;
                        } else {
                        } else
                                ix += ix;
                        }

#endif /* ! codereview */
                        m += m;
                        iz = ix - iy;

#endif /* ! codereview */
                        if (iz >= 0) {
                                m += 1;
                                ix = iz + iz;
                        } else {
                        } else
                                ix += ix;
                        }

#endif /* ! codereview */
                        m += m;
                        iz = ix - iy;
```



```
#endif /* ! codereview */
                        if (iz >= 0) {
                                m += 1;
                                ix = iz + iz;
                        } else {
                        } else
                                ix += ix;
                        }

#endif /* ! codereview */
                        m += m;
                        iz = ix - iy;

#endif /* ! codereview */
                        if (iz >= 0) {
                                m += 1;
                                ix = iz + iz;
                        } else {
                        } else
                                ix += ix;
                        }

#endif /* ! codereview */
                        m += m;

#endif /* ! codereview */
                        if (iz == 0) {
                                iz = (k << 2) + nd;

#endif /* ! codereview */
                                if (iz < 32)
                                        m <<= iz;
                                else
                                        m = 0;

#endif /* ! codereview */
                                m &= 0x7fffffff;
                                *quo = sq >= 0 ? m : -m;
                                *(int *)&w = is & hx;
                                *(int *) &w = is & hx;
                                return (w);
                        }
                }

#endif /* ! codereview */
                while (nd--) {
                        iz = ix - iy;

#endif /* ! codereview */
                        if (iz >= 0) {
                                m += 1;
                                ix = iz + iz;
                        } else {
                        } else
                                ix += ix;
                        }

#endif /* ! codereview */
                        m += m;
                }

#endif /* ! codereview */
                /* end of unrolling */

                iz = ix - iy;
```

```
#endif /* ! codereview */
                if (iz >= 0) {
                        m += 1;
                        ix = iz;
                }

#endif /* ! codereview */
                m &= 0x7fffffff;
                *quo = sq >= 0 ? m : -m;

                /* convert back to floating value and restore the sign */
                if (ix == 0) {
                        *(int *)&w = is & hx;
                        *(int *) &w = is & hx;
                        return (w);
                }

#endif /* ! codereview */
                while (ix < iu) {
                        ix += ix;
                        ny -= 1;
                }

#endif /* ! codereview */
                while (ix > (iu + iu)) {
                        ny += 1;
                        ix >>= 1;
                }

                if (ny > 0) {
                        *(int *)&w = (is & hx) | (ix & im) | (ny << 23);
                } else {                /* subnormal output */
                if (ny > 0)
                        *(int *) &w = (is & hx) | (ix & im) | (ny << 23);
                else {          /* subnormal output */
                        k = -ny + 1;
                        ix >>= k;
                        *(int *)&w = (is & hx) | ix;
                        *(int *) &w = (is & hx) | ix;
                }
        }

#endif /* ! codereview */
        return (w);
}

float
remquof(float x, float y, int *quo)
{
remquof(float x, float y, int *quo) {
        int hx, hy, sx, sq;
        float v;

        hx = *(int *)&x;                /* high word of x */
        hy = *(int *)&y;                /* high word of y */
        hx = *(int *) &x;       /* high word of x */
        hy = *(int *) &y;       /* high word of y */
        sx = hx & is;                   /* sign of x */
        sq = (hx ^ hy) & is;            /* sign of x/y */
        hx ^= sx;                       /* |x| */
        hy &= 0x7fffffff;               /* |y| */

        /* purge off exception values: y is 0 or NaN, x is Inf or NaN */
        *quo = 0;
```



```
#endif /* ! codereview */
        if (hx >= ii || hy > ii || hy == 0) {
                v = x * y;
                return (v / v);
        }

        y = fabsf(y);
        x = fabsf(x);

#endif /* ! codereview */
        if (hy <= 0x7f7fffff) {
                x = fmodquof(x, y + y, quo);
                *quo = ((*quo) & 0x3fffffff) << 1;
        }

#endif /* ! codereview */
        if (hy < 0x01000000) {
                if (x + x > y) {
                        *quo += 1;

#endif /* ! codereview */
                        if (x == y)
                                x = zero;
                        else
                                x -= y;

#endif /* ! codereview */
                        if (x + x >= y) {
                                x -= y;
                                *quo += 1;
                        }
                }
        } else {
                v = half * y;

#endif /* ! codereview */
                if (x > v) {
                        *quo += 1;

#endif /* ! codereview */
                        if (x == y)
                                x = zero;
                        else
                                x -= y;

#endif /* ! codereview */
                        if (x >= v) {
                                x -= y;
                                *quo += 1;
                        }
                }
        }

#endif /* ! codereview */
        if (sq != 0)
                *quo = -(*quo);

#endif /* ! codereview */
        return (sx == 0 ? x : -x);
}
```

```
**********************************************************
    6860 Thu Jun  6 20:40:34 2019
new/usr/src/lib/libm/common/m9x/remquol.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __remquol = remquol

#include "libm.h"
#if defined(__SUNPRO_C)
#include <sunmath.h>                    /* fabsl */
#endif
/* INDENT OFF */
static const int
        is = -0x7fffffff - 1,
        im = 0x0000ffff,
        iu = 0x00010000;

static const int is = -0x7fffffff - 1, im = 0x0000ffff, iu = 0x00010000;
#endif /* ! codereview */
static const long double zero = 0.0L, one = 1.0L;

/* INDENT ON */

#if defined(__sparc)
#define __H0(x)         ((int *)&x)[0]
#define __H1(x)         ((int *)&x)[1]
#define __H2(x)         ((int *)&x)[2]
#define __H3(x)         ((int *)&x)[3]
#define __H0(x) ((int *) &x)[0]
#define __H1(x) ((int *) &x)[1]
#define __H2(x) ((int *) &x)[2]
#define __H3(x) ((int *) &x)[3]
#else
#error Unsupported architecture
#endif
```



```
/*
 * On entrance: *quo is initialized to 0, x finite and y non-zero & ordered
 */
static long double
fmodquol(long double x, long double y, int *quo)
{
fmodquol(long double x, long double y, int *quo) {
        long double a, b;
        int n, ix, iy, k, sx, sq, m;
        int hx;
        int x0, y0, z0, carry;
        unsigned x1, x2, x3, y1, y2, y3, z1, z2, z3;

        hx = __H0(x);
        x1 = __H1(x);
        x2 = __H2(x);
        x3 = __H3(x);
        y0 = __H0(y);
        y1 = __H1(y);
        y2 = __H2(y);
        y3 = __H3(y);

        sx = hx & is;
        sq = (hx ^ y0) & is;
        x0 = hx ^ sx;
        y0 &= ~0x80000000;

        a = fabsl(x);
        b = fabsl(y);

#endif /* ! codereview */
        if (a <= b) {
                if (a < b) {
                if (a < b)
                        return (x);
                } else {
                else {
                        *quo = 1 + (sq >> 30);
                        return (zero * x);
                }
        }

#endif /* ! codereview */
        /* determine ix = ilogbl(x) */
        if (x0 < iu) {                  /* subnormal x */
                ix = 0;
                ix = -16382;

#endif /* ! codereview */
                while (x0 == 0) {
                        ix -= 16;
                        x0 = x1 >> 16;
                        x1 = (x1 << 16) | (x2 >> 16);
                        x2 = (x2 << 16) | (x3 >> 16);
                        x3 = (x3 << 16);
                }

#endif /* ! codereview */
                while (x0 < iu) {
                        ix -= 1;
                        x0 = (x0 << 1) | (x1 >> 31);
                        x1 = (x1 << 1) | (x2 >> 31);
                        x2 = (x2 << 1) | (x3 >> 31);
                        x3 <<= 1;
                }
```

```
        } else {
                ix = (x0 >> 16) - 16383;
                x0 = iu | (x0 & im);
        }

        /* determine iy = ilogbl(y) */
        if (y0 < iu) {                  /* subnormal y */
                iy = -16382;

#endif /* ! codereview */
                while (y0 == 0) {
                        iy -= 16;
                        y0 = y1 >> 16;
                        y1 = (y1 << 16) | (y2 >> 16);
                        y2 = (y2 << 16) | (y3 >> 16);
                        y3 = (y3 << 16);
                }

#endif /* ! codereview */
                while (y0 < iu) {
                        iy -= 1;
                        y0 = (y0 << 1) | (y1 >> 31);
                        y1 = (y1 << 1) | (y2 >> 31);
                        y2 = (y2 << 1) | (y3 >> 31);
                        y3 <<= 1;
                }
        } else {
                iy = (y0 >> 16) - 16383;
                y0 = iu | (y0 & im);
        }


        /* fix point fmod */
        n = ix - iy;
        m = 0;

#endif /* ! codereview */
        while (n--) {
                while (x0 == 0 && n >= 16) {
                        m <<= 16;
                        n -= 16;
                        x0 = x1 >> 16;
                        x1 = (x1 << 16) | (x2 >> 16);
                        x2 = (x2 << 16) | (x3 >> 16);
                        x3 = (x3 << 16);
                }

#endif /* ! codereview */
                while (x0 < iu && n >= 1) {
                        m += m;
                        n -= 1;
                        x0 = (x0 << 1) | (x1 >> 31);
                        x1 = (x1 << 1) | (x2 >> 31);
                        x2 = (x2 << 1) | (x3 >> 31);
                        x3 = (x3 << 1);
                }

#endif /* ! codereview */
                carry = 0;
                z3 = x3 - y3;
                carry = z3 > x3;

#endif /* ! codereview */
                if (carry == 0) {
                        z2 = x2 - y2;
                        carry = z2 > x2;
```



```
                } else {
                        z2 = x2 - y2 - 1;
                        carry = z2 >= x2;
                }

#endif /* ! codereview */
                if (carry == 0) {
                        z1 = x1 - y1;
                        carry = z1 > x1;
                } else {
                        z1 = x1 - y1 - 1;
                        carry = z1 >= x1;
                }

#endif /* ! codereview */
                z0 = x0 - y0 - carry;

#endif /* ! codereview */
                if (z0 < 0) {           /* double x */
                        x0 = x0 + x0 + ((x1 & is) != 0);
                        x1 = x1 + x1 + ((x2 & is) != 0);
                        x2 = x2 + x2 + ((x3 & is) != 0);
                        x3 = x3 + x3;
                        m += m;
                } else {
                        m += 1;

#endif /* ! codereview */
                        if (z0 == 0) {
                                if ((z1 | z2 | z3) == 0) {
                                        /* 0: we are done */
                                        if (n < 31)
                                                m <<= (1 + n);
                                        else
                                                m = 0;

#endif /* ! codereview */
                                        m &= ~0x80000000;
                                        *quo = sq >= 0 ? m : -m;
                                        __H0(a) = hx & is;
                                        __H1(a) = __H2(a) = __H3(a) = 0;
                                        return (a);
                                }
                        }

#endif /* ! codereview */
                        /* x = z << 1 */
                        z0 = z0 + z0 + ((z1 & is) != 0);
                        z1 = z1 + z1 + ((z2 & is) != 0);
                        z2 = z2 + z2 + ((z3 & is) != 0);
                        z3 = z3 + z3;
                        x0 = z0;
                        x1 = z1;
                        x2 = z2;
                        x3 = z3;
                        m += m;
                }
        }

#endif /* ! codereview */
        carry = 0;
        z3 = x3 - y3;
        carry = z3 > x3;

#endif /* ! codereview */
        if (carry == 0) {
```

```
                z2 = x2 - y2;
                carry = z2 > x2;
        } else {
                z2 = x2 - y2 - 1;
                carry = z2 >= x2;
        }

#endif /* ! codereview */
        if (carry == 0) {
                z1 = x1 - y1;
                carry = z1 > x1;
        } else {
                z1 = x1 - y1 - 1;
                carry = z1 >= x1;
        }

#endif /* ! codereview */
        z0 = x0 - y0 - carry;

#endif /* ! codereview */
        if (z0 >= 0) {
                x0 = z0;
                x1 = z1;
                x2 = z2;
                x3 = z3;
                m += 1;
        }

#endif /* ! codereview */
        m &= ~0x80000000;
        *quo = sq >= 0 ? m : -m;

        /* convert back to floating value and restore the sign */
        if ((x0 | x1 | x2 | x3) == 0) {
                __H0(a) = hx & is;
                __H1(a) = __H2(a) = __H3(a) = 0;
                return (a);
        }

#endif /* ! codereview */
        while (x0 < iu) {
                if (x0 == 0) {
                        iy -= 16;
                        x0 = x1 >> 16;
                        x1 = (x1 << 16) | (x2 >> 16);
                        x2 = (x2 << 16) | (x3 >> 16);
                        x3 = (x3 << 16);
                } else {
                        x0 = x0 + x0 + ((x1 & is) != 0);
                        x1 = x1 + x1 + ((x2 & is) != 0);
                        x2 = x2 + x2 + ((x3 & is) != 0);
                        x3 = x3 + x3;
                        iy -= 1;
                }
        }

        /* normalize output */
        if (iy >= -16382) {
                __H0(a) = sx | (x0 - iu) | ((iy + 16383) << 16);
                __H1(a) = x1;
                __H2(a) = x2;
                __H3(a) = x3;
        } else {                        /* subnormal output */
                n = -16382 - iy;
                k = n & 31;
```



```
#endif /* ! codereview */
                if (k <= 16) {
                        x3 = (x2 << (32 - k)) | (x3 >> k);
                        x2 = (x1 << (32 - k)) | (x2 >> k);
                        x1 = (x0 << (32 - k)) | (x1 >> k);
                        x0 >>= k;
                } else {
                        x3 = (x2 << (32 - k)) | (x3 >> k);
                        x2 = (x1 << (32 - k)) | (x2 >> k);
                        x1 = (x0 << (32 - k)) | (x1 >> k);
                        x0 = 0;
                }

#endif /* ! codereview */
                while (n >= 32) {
                        n -= 32;
                        x3 = x2;
                        x2 = x1;
                        x1 = x0;
                        x0 = 0;
                }

#endif /* ! codereview */
                __H0(a) = x0 | sx;
                __H1(a) = x1;
                __H2(a) = x2;
                __H3(a) = x3;
                a *= one;
        }

#endif /* ! codereview */
        return (a);
}

long double
remquol(long double x, long double y, int *quo)
{
 80 remquol(long double x, long double y, int *quo) {
        int hx, hy, sx, sq;
        long double v;

        hx = __H0(x);                   /* high word of x */
        hy = __H0(y);                   /* high word of y */
        sx = hx & is;                   /* sign of x */
        sq = (hx ^ hy) & is;            /* sign of x/y */
        hx ^= sx;                       /* |x| */
        hy &= ~0x80000000;

        /* purge off exception values */
        *quo = 0;

#endif /* ! codereview */
        /* y=0, y is NaN, x is NaN or inf */
        if (y == 0.0L || y != y || hx >= 0x7fff0000)
                return ((x * y) / (x * y));

        y = fabsl(y);
        x = fabsl(x);

#endif /* ! codereview */
        if (hy <= 0x7ffdffff) {
                x = fmodquol(x, y + y, quo);
                *quo = ((*quo) & 0x3fffffff) << 1;
        }

#endif /* ! codereview */
```

```
        if (hy < 0x00020000) {
                if (x + x > y) {
                        *quo += 1;

#endif /* ! codereview */
                        if (x == y)
                                x = zero;
                        else
                                x -= y;

#endif /* ! codereview */
                        if (x + x >= y) {
                                x -= y;
                                *quo += 1;
                        }
                }
        } else {
                v = 0.5L * y;

#endif /* ! codereview */
                if (x > v) {
                        *quo += 1;

#endif /* ! codereview */
                        if (x == y)
                                x = zero;
                        else
                                x -= y;

#endif /* ! codereview */
                        if (x >= v) {
                                x -= y;
                                *quo += 1;
                        }
                }
        }

#endif /* ! codereview */
        if (sq != 0)
                *quo = -(*quo);

#endif /* ! codereview */
        return (sx == 0 ? x : -x);
}
```

```
**********************************************************
    1938 Thu Jun  6 20:40:34 2019
new/usr/src/lib/libm/common/m9x/round.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __round = round

#include "libm.h"

double
round(double x)
{
round(double x) {
        union {
                unsigned i[2];
                double d;
        } xx;

#endif /* ! codereview */
        unsigned hx, sx, i;

        xx.d = x;
        hx = xx.i[HIWORD] & ~0x80000000;
        sx = xx.i[HIWORD] & 0x80000000;

#endif /* ! codereview */
        if (hx < 0x43300000) {          /* |x| < 2^52 */
                if (hx < 0x3ff00000) {  /* |x| < 1 */
                        if (hx >= 0x3fe00000)
                                return (sx ? -1.0 : 1.0);

#endif /* ! codereview */
                        return (sx ? -0.0 : 0.0);
                }
```



```
                /* round x at the integer bit */
                if (hx < 0x41300000) {
                        i = 1 << (0x412 - (hx >> 20));
                        xx.i[HIWORD] = (xx.i[HIWORD] + i) & ~(i | (i - 1));
                        xx.i[LOWORD] = 0;
                } else {
                        i = 1 << (0x432 - (hx >> 20));
                        xx.i[LOWORD] += i;

#endif /* ! codereview */
                        if (xx.i[LOWORD] < i)
                                xx.i[HIWORD]++;

#endif /* ! codereview */
                        xx.i[LOWORD] &= ~(i | (i - 1));
                }

#endif /* ! codereview */
                return (xx.d);
        } else if (hx < 0x7ff00000) {
        } else if (hx < 0x7ff00000)
                return (x);
        } else {
        else
#if defined(FPADD_TRAPS_INCOMPLETE_ON_NAN)
                return (hx >= 0x7ff80000 ? x : x + x);

#endif /* ! codereview */
        /* assumes sparc-like QNaN */
#else
                return (x + x);
#endif
        }
#endif /* ! codereview */
}
```

```
**********************************************************
    1698 Thu Jun  6 20:40:35 2019
new/usr/src/lib/libm/common/m9x/roundf.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __roundf = roundf

#include "libm.h"

float
roundf(float x)
{
roundf(float x) {
        union {
                unsigned i;
                float f;
        } xx;

#endif /* ! codereview */
        unsigned hx, sx, i;

        xx.f = x;
        hx = xx.i & ~0x80000000;
        sx = xx.i & 0x80000000;

#endif /* ! codereview */
        if (hx < 0x4b000000) {          /* |x| < 2^23 */
                if (hx < 0x3f800000) {  /* |x| < 1 */
                        if (hx >= 0x3f000000)
                                return (sx ? -1.0F : 1.0F);

#endif /* ! codereview */
                        return (sx ? -0.0F : 0.0F);
                }
```



```
                /* round x at the integer bit */
                i = 1 << (0x95 - (hx >> 23));
                xx.i = (xx.i + i) & ~((i << 1) - 1);
                return (xx.f);
        } else if (hx < 0x7f800000) {   /* |x| is integral */
        } else if (hx < 0x7f800000)     /* |x| is integral */
                return (x);
        } else {
        else
#if defined(FPADD_TRAPS_INCOMPLETE_ON_NAN)
                return (hx > 0x7f800000 ? x * x : x + x);
#else
                return (x + x);
#endif
        }
#endif /* ! codereview */
}
```

```
**********************************************************
    3689 Thu Jun  6 20:40:35 2019
new/usr/src/lib/libm/common/m9x/roundl.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __roundl = roundl

#include "libm.h"

#if defined(__sparc)
long double
roundl(long double x)
{
roundl(long double x) {
        union {
                unsigned i[4];
                long double q;
        } xx;

#endif /* ! codereview */
        unsigned hx, sx, v;
        int j;

        xx.q = x;
        sx = xx.i[0] & 0x80000000;
        hx = xx.i[0] & ~0x80000000;

        /* handle trivial cases */
        if (hx >= 0x406f0000)           /* |x| >= 2^112 + ... or x is nan */
                return (hx >= 0x7fff0000 ? x + x : x);

        /* handle |x| < 1 */
        if (hx < 0x3fff0000) {
                if (hx >= 0x3ffe0000)
                        return (sx ? -1.0L : 1.0L);
```



```
#endif /* ! codereview */
                return (sx ? -0.0L : 0.0L);
        }

        xx.i[0] = hx;
        j = 0x406f - (hx >> 16);        /* 1 <= j <= 112 */

#endif /* ! codereview */
        if (j >= 96) {                  /* 96 <= j <= 112 */
                v = (1U << (j - 96)) >> 1;

#endif /* ! codereview */
                if (v) {
                        if (xx.i[0] & v)
                                xx.i[0] += v;

#endif /* ! codereview */
                        xx.i[0] &= ~(v - 1);
                } else if (xx.i[1] & 0x80000000) {
                } else if (xx.i[1] & 0x80000000)
                        ++xx.i[0];
                }

#endif /* ! codereview */
                xx.i[1] = xx.i[2] = xx.i[3] = 0;
        } else if (j >= 64) {           /* 64 <= j <= 95 */
                v = (1U << (j - 64)) >> 1;

#endif /* ! codereview */
                if (v) {
                        if (xx.i[1] & v) {
                                xx.i[1] += v;

#endif /* ! codereview */
                                if (xx.i[1] < v)
                                        ++xx.i[0];
                        }

#endif /* ! codereview */
                        xx.i[1] &= ~(v - 1);
                } else if (xx.i[2] & 0x80000000) {
                        if (++xx.i[1] == 0)
                                ++xx.i[0];
                }

#endif /* ! codereview */
                xx.i[2] = xx.i[3] = 0;
        } else if (j >= 32) {           /* 32 <= j <= 63 */
                v = (1U << (j - 32)) >> 1;

#endif /* ! codereview */
                if (v) {
                        if (xx.i[2] & v) {
                                xx.i[2] += v;

#endif /* ! codereview */
                                if (xx.i[2] < v) {
                                        if (++xx.i[1] == 0)
                                                ++xx.i[0];
                                }
                        }

#endif /* ! codereview */
                        xx.i[2] &= ~(v - 1);
                } else if (xx.i[3] & 0x80000000) {
```

```
                        if (++xx.i[2] == 0) {
                                if (++xx.i[1] == 0)
                                        ++xx.i[0];
                        }
                }

#endif /* ! codereview */
                xx.i[3] = 0;
        } else {                        /* 1 <= j <= 31 */
                v = 1U << (j - 1);

#endif /* ! codereview */
                if (xx.i[3] & v) {
                        xx.i[3] += v;

#endif /* ! codereview */
                        if (xx.i[3] < v) {
                                if (++xx.i[2] == 0) {
                                        if (++xx.i[1] == 0)
                                                ++xx.i[0];
                                }
                        }
                }

#endif /* ! codereview */
                xx.i[3] &= ~(v - 1);
        }

        /* negate result if need be */
        if (sx)
                xx.i[0] |= 0x80000000;

#endif /* ! codereview */
        return (xx.q);
}
#elif defined(__x86)
long double
roundl(long double x)
{
roundl(long double x) {
        union {
                unsigned i[3];
                long double e;
        } xx;

#endif /* ! codereview */
        int ex, sx, i;

        xx.e = x;
        ex = xx.i[2] & 0x7fff;
        sx = xx.i[2] & 0x8000;

#endif /* ! codereview */
        if (ex < 0x403e) {              /* |x| < 2^63 */
                if (ex < 0x3fff) {      /* |x| < 1 */
                        if (ex >= 0x3ffe)
                                return (sx ? -1.0L : 1.0L);

#endif /* ! codereview */
                        return (sx ? -0.0L : 0.0L);
                }

                /* round x at the integer bit */
                if (ex < 0x401e) {
                        i = 1 << (0x401d - ex);
                        xx.i[1] = (xx.i[1] + i) & ~(i | (i - 1));
```



```
                        xx.i[0] = 0;
                } else {
                        i = 1 << (0x403d - ex);
                        xx.i[0] += i;

#endif /* ! codereview */
                        if (xx.i[0] < i)
                                xx.i[1]++;

#endif /* ! codereview */
                        xx.i[0] &= ~(i | (i - 1));
                }

#endif /* ! codereview */
                if (xx.i[1] == 0) {
                        xx.i[2] = sx | ++ex;
                        xx.i[1] = 0x80000000U;
                }

#endif /* ! codereview */
                return (xx.e);
        } else if (ex < 0x7fff) {       /* x is integral */
        } else if (ex < 0x7fff) /* x is integral */
                return (x);
        } else {                        /* inf or nan */
        else                    /* inf or nan */
                return (x + x);
        }
#endif /* ! codereview */
}
#else
#error Unknown architecture
#endif /* defined(__sparc) || defined(__x86) */
```

```
**********************************************************
    2673 Thu Jun  6 20:40:36 2019
new/usr/src/lib/libm/common/m9x/scalbln.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __scalbln = scalbln

#include "libm.h"
#include <float.h>                      /* DBL_MAX, DBL_MIN */

static const double twom54 = 5.551115123125782702118158340454101562 5e-17;

#endif /* ! codereview */
#if defined(__x86)
static const double two52 = 4503599627370496.0;
#else
/*
 * Normalize non-zero subnormal x and return biased exponent of x in [-51,0]
 */
static int
ilogb_biased(unsigned *px)
{
ilogb_biased(unsigned *px) {
        int s = 52;
        unsigned v = px[HIWORD] & ~0x80000000, w = px[LOWORD], t = v;

        if (t)
                s -= 32;
        else
                t = w;

#endif /* ! codereview */
        if (t & 0xffff0000)
                s -= 16, t >>= 16;
```



```
#endif /* ! codereview */
        if (t & 0xff00)
                s -= 8, t >>= 8;

#endif /* ! codereview */
        if (t & 0xf0)
                s -= 4, t >>= 4;

#endif /* ! codereview */
        t <<= 1;
        s -= (0xffffaa50 >> t) & 0x3;

#endif /* ! codereview */
        if (s < 32) {
                v = (v << s) | w >> (32 - s);
                w <<= s;
        } else {
                v = w << (s - 32);
                w = 0;
        }

#endif /* ! codereview */
        px[HIWORD] = (px[HIWORD] & 0x80000000) | v;
        px[LOWORD] = w;
        return (1 - s);
}
#endif /* defined(__x86) */

double
scalbln(double x, long n)
{
        int *px = (int *)&x, ix, k;
scalbln(double x, long n) {
        int *px = (int *) &x, ix, k;

        ix = px[HIWORD] & ~0x80000000;
        k = ix >> 20;

#endif /* ! codereview */
        if (k == 0x7ff)
#if defined(FPADD_TRAPS_INCOMPLETE_ON_NAN)
                return ((px[HIWORD] & 0x80000) != 0 ? x : x + x);

#endif /* ! codereview */
        /* assumes sparc-like QNaN */
#else
                return (x + x);
#endif

#endif /* ! codereview */
        if ((px[LOWORD] | ix) == 0 || n == 0)
                return (x);

#endif /* ! codereview */
        if (k == 0) {
#if defined(__x86)
                x *= two52;
                k = ((px[HIWORD] & ~0x80000000) >> 20) - 52;
#else
                k = ilogb_biased((unsigned *)px);
                k = ilogb_biased((unsigned *) px);
#endif
        }

        k += (int)n;
```

```
        k += (int) n;
        if (n > 5000 || k > 0x7fe)
                return (DBL_MAX * copysign(DBL_MAX, x));

#endif /* ! codereview */
        if (n < -5000 || k <= -54)
                return (DBL_MIN * copysign(DBL_MIN, x));

#endif /* ! codereview */
        if (k > 0) {
                px[HIWORD] = (px[HIWORD] & ~0x7ff00000) | (k << 20);
                return (x);
        }

#endif /* ! codereview */
        k += 54;
        px[HIWORD] = (px[HIWORD] & ~0x7ff00000) | (k << 20);
        return (x * twom54);
}
```

```
**********************************************************
    2285 Thu Jun  6 20:40:36 2019
new/usr/src/lib/libm/common/m9x/scalblnf.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __scalblnf = scalblnf

#include "libm.h"
#include <float.h>                      /* FLT_MAX, FLT_MIN */

static const float twom25f = 2.98023223876953125e-8F;

#endif /* ! codereview */
#if defined(__x86)
static const float two23f = 8388608.0F;
#else
/*
 * v: a non-zero subnormal |x|; returns [-22, 0]
 */
static int
ilogbf_biased(unsigned v)
{
ilogbf_biased(unsigned v) {
        int r = -22;

        if (v & 0xffff0000)
                r += 16, v >>= 16;

#endif /* ! codereview */
        if (v & 0xff00)
                r += 8, v >>= 8;

#endif /* ! codereview */
        if (v & 0xf0)
                r += 4, v >>= 4;
```



```
#endif /* ! codereview */
        v <<= 1;
        return (r + ((0xffffaa50 >> v) & 0x3));
}
#endif /* defined(__x86) */

float
scalblnf(float x, long n)
{
        int *px = (int *)&x, ix, k;
scalblnf(float x, long n) {
        int *px = (int *) &x, ix, k;

        ix = *px & ~0x80000000;
        k = ix >> 23;

#endif /* ! codereview */
        if (k == 0xff)
#if defined(FPADD_TRAPS_INCOMPLETE_ON_NAN)
                return (ix > 0x7f800000 ? x * x : x);
#else
                return (x + x);
#endif

#endif /* ! codereview */
        if (ix == 0 || n == 0)
                return (x);

#endif /* ! codereview */
        if (k == 0) {
#if defined(__x86)
                x *= two23f;
                k = ((*px & ~0x80000000) >> 23) - 23;
#else
                k = ilogbf_biased(ix);
                *px = (*px & 0x80000000) | (ix << (-k + 1));
#endif
        }

        k += (int)n;

        k += (int) n;
        if (n > 5000 || k > 0xfe)
                return (FLT_MAX * copysignf(FLT_MAX, x));

#endif /* ! codereview */
        if (n < -5000 || k <= -25)
                return (FLT_MIN * copysignf(FLT_MIN, x));

#endif /* ! codereview */
        if (k > 0) {
                *px = (*px & ~0x7f800000) | (k << 23);
                return (x);
        }

#endif /* ! codereview */
        k += 25;
        *px = (*px & ~0x7f800000) | (k << 23);
        return (x * twom25f);
}
```

```
**********************************************************
    2395 Thu Jun  6 20:40:37 2019
new/usr/src/lib/libm/common/m9x/scalblnl.c
11210 libm should be cstyle(10NBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __scalblnl = scalblnl

#include "libm.h"
#include <float.h>                      /* LDBL_MAX, LDBL_MIN */

#if defined(__sparc)
#define XSET_EXP(k, x)                  ((int *)&x)[0] = (((int *)&x)[0] & \
            ~0x7fff0000) | (k << 16)
#define XSET_EXP(k, x)  ((int *) &x)[0] = (((int *) &x)[0] & ~0x7fff0000) | \
                                (k << 16)
#define ISINFNANL(k, x)                 (k == 0x7fff)
#define XTWOT_OFFSET                    113

static const long double
        xtwot = 10384593717069655257060992658440192.0L, /* 2^113 */
static const long double xtwot = 10384593717069655257060992658440192.0L,
                                                        /* 2^113 */
        twomtm1 = 4.814824860968089632639944856462318296E-35L;  /* 2^-114 */
#elif defined(__x86)
#define XSET_EXP(k, x)                  ((int *)&x)[2] = (((int *)&x)[2] & \
            ~0x7fff) | k
#define XSET_EXP(k, x)  ((int *) &x)[2] = (((int *) &x)[2] & ~0x7fff) | k
#if defined(HANDLE_UNSUPPORTED)
#define ISINFNANL(k, x)         (k == 0x7fff || (k != 0 && \
        (((int *)&x)[1] & 0x80000000) == 0))
#define      ISINFNANL(k, x) (k == 0x7fff || \
                        (k != 0 && (((int *) &x)[1] & 0x80000000) == 0))
#else
#define ISINFNANL(k, x)                 (k == 0x7fff)
#endif
```



```
#define XTWOT_OFFSET                    64

#endif /* ! codereview */
static const long double xtwot = 18446744073709551616.0L,       /* 2^64 */
        twomtm1 = 2.710505431213761085018 6E-20L;               /* 2^-65 */
#endif

long double
scalblnl(long double x, long n)
{
scalblnl(long double x, long n) {
        int k = XBIASED_EXP(x);

        if (ISINFNANL(k, x))
                return (x + x);

#endif /* ! codereview */
        if (ISZEROL(x) || n == 0)
                return (x);

#endif /* ! codereview */
        if (k == 0) {
                x *= xtwot;
                k = XBIASED_EXP(x) - XTWOT_OFFSET;
        }

        k += (int)n;

        k += (int) n;
        if (n > 50000 || k > 0x7ffe)
                return (LDBL_MAX * copysignl(LDBL_MAX, x));

#endif /* ! codereview */
        if (n < -50000 || k <= -XTWOT_OFFSET - 1)
                return (LDBL_MIN * copysignl(LDBL_MIN, x));

#endif /* ! codereview */
        if (k > 0) {
                XSET_EXP(k, x);
                return (x);
        }

#endif /* ! codereview */
        k += XTWOT_OFFSET + 1;
        XSET_EXP(k, x);
        return (x * twomtm1);
}
```

```
**********************************************************
   68831 Thu Jun  6 20:40:37 2019
new/usr/src/lib/libm/common/m9x/tgamma.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __tgamma = tgamma

/* BEGIN CSTYLED */
/* INDENT OFF */
/*
 * True gamma function
 * double tgamma(double x)
 *
 * Error:
 * ------
 *      Less that one ulp for both positive and negative arguments.
 *
 * Algorithm:
 * ---------
 *      A: For negative argument
 *              (1) gamma(-n or -inf) is NaN
 *              (2) Underflow Threshold
 *              (3) Reduction to gamma(1+x)
 *      B: For x between 1 and 2
 *      C: For x between 0 and 1
 *      D: For x between 2 and 8
 *      E: Overflow thresold {see over.c}
 *      F: For overflow_threshold >= x >= 8
 *
 * Implementation details
 * ----------------------
 *                                                        -pi
 * (A) For negative argument, use gamma(-x) = ------------------------.
 *                                            (sin(pi*x)*gamma(1+x))
 *
```



```
 *   (1) gamma(-n or -inf) is NaN with invalid signal by SUSv3 spec.
 *       (Ideally, gamma(-n) = 1/sinpi(n) = (-1)**(n+1) * inf.)
 *
 *   (2) Underflow Threshold. For each precision, there is a value T
 *      such that when x>T and when x is not an integer, gamma(-x) will
 *       always underflow. A table of the underflow threshold value is given
 *      below. For proof, see file "under.c".
 *
 *      Precision       underflow threshold T =
 *      ---------------------------------------------------------------------
 *      single  41.000041962                                    = 41  + 11 ULP
 *              (machine format) 4224000B
 *      double  183.00000000000312639                           = 183 + 11 ULP
 *              (machine format) 4066E000 0000000B
 *      quad    1774.00000000000000000000000000017749370        = 1774 + 9 ULP
 *              (machine format) 4009BB8000000000000000000000009
 *      ---------------------------------------------------------------------
 *
 *   (3) Reduction to gamma(1+x).
 *      Because of (1) and (2), we need only consider non-integral x
 *      such that 0<x<T. Let k = [x] and z = x-[x]. Define
 *                      sin(x*pi)                cos(x*pi)
 *      kpsin(x) = --------- and kpcos(x) = --------- . Then
 *                         pi                       pi
 *                                      1
 *              gamma(-x) = --------------------.
 *                                -kpsin(x)*gamma(1+x)
 *      Since x = k+z,
 *                                                      k+1
 *              -sin(x*pi) = -sin(k*pi+z*pi) = (-1)   *sin(z*pi),
 *                                     k+1
 *      we have -kpsin(x) = (-1)   * kpsin(z).  We can further
 *      reduce z to t by
 *         (I)   t = z       when 0.00000     <= z < 0.31830...
 *         (II)  t = 0.5-z   when 0.31830...  <= z < 0.681690...
 *         (III) t = 1-z     when 0.681690... <= z < 1.00000
 *      and correspondingly
 *         (I)   kpsin(z) = kpsin(t)    ... 0<= z < 0.3184
 *         (II)  kpsin(z) = kpcos(t)    ... |t|   < 0.182
 *         (III) kpsin(z) = kpsin(t)    ... 0<= t < 0.3184
 *
 *      Using a special Remez algorithm, we obtain the following polynomial
 *      approximation for kpsin(t) for 0<=t<0.3184:
 *
 *      Computation note: in simulating higher precision arithmetic, kcpsin
 *      return head = t and tail = ks[0]*t^3 + (...) to maintain extra bits.
 *
 *      Quad precision, remez error <= 2**(-129.74)
 *                                      3            5                   27
 *          kpsin(t) = t + ks[0] * t  + ks[1] * t  + ... + ks[12] * t
 *
 *       ks[ 0] =  -1.6449340668482264364724151666460251870515890287 0e+0000
 *       ks[ 1] =   8.1174242528335364363700277240587423809499572616 0e-0001
 *       ks[ 2] =  -1.9075182412208421369647211183533736623228272393 3e-0001
 *       ks[ 3] =   2.6147847817654800504653261356324128811539551708 4e-0002
 *       ks[ 4] =  -2.3460810354558236375089307264711782944801647997 1e-0003
 *       ks[ 5] =   1.4842879303107100368460664721253402755626204015 8e-0004
 *       ks[ 6] =  -6.9758736616563804651846272225276812261595289869 8e-0006
 *       ks[ 7] =   2.5312174041370253692865927174718750093484005792 9e-0007
 *       ks[ 8] =  -7.3047118222138599039768364169576612130193362195 6e-0009
 *       ks[ 9] =   1.7165384745116349573995824969554931398797358988 4e-0010
 *       ks[10] =  -3.3481331471456077612224579692905481345834142056 5e-0012
 *       ks[11] =   5.5072499226262203344948780830696913543141175304 7e-0014
 *       ks[12] =  -7.6767813275357799860123439321580222110423697992 8e-0016
 *
 *      Double precision, Remez error <= 2**(-62.9)
```

```
 *                                 3            5                  15
 *          kpsin(t) = t + ks[0] * t  + ks[1] * t  + ... + ks[6] * t
 *
 *       ks[0] =  -1.6449340668482264060656 91   (0x3ffa51a6 625307d3)
 *       ks[1] =   8.1174242528334165588366874187400892085069859062 1e-0001
 *       ks[2] =  -1.9075182412086287382559727911830494399404225829 1e-0001
 *       ks[3] =   2.6147847763255427831728962833265453935352191157 0e-0002
 *       ks[4] =  -2.3460797851020271037761719027873552535434770586 6e-0003
 *       ks[5] =   1.4841329229005169589724289997712184676382422170 5e-0004
 *       ks[6] =  -6.8773076963754348810868872677687262485357072242 e-0006
 *
 *      Single precision, Remez error <= 2**(-34.09)
 *                                 3            5                  9
 *          kpsin(t) = t + ks[0] * t  + ks[1] * t  + ... + ks[3] * t
 *
 *       ks[0] =  -1.6449340498564581135447666505200534283944779054 4e+0000
 *       ks[1] =   8.1174079445835106409279724906943826936738927227 0e-0001
 *       ks[2] =  -1.9070314460355121693307580916288953687885405520 2e-0001
 *       ks[3] =   2.5574233399426456328115531227148110863557533120 1e-0002
 *
 *      Computation note: in simulating higher precision arithmetic, kcpsin
 *      return head = t and tail = kc[0]*t^3 + (...) to maintain extra bits
 *      precision.
 *
 *      And for kpcos(t) for |t|< 0.183:
 *
 *      Quad precision, remez <= 2**(-122.48)
 *                                 2            4                  22
 *          kpcos(t) = 1/pi +  pi/2 * t  + kc[2] * t + ... + kc[11] * t
 *
 *       kc[2] =   1.2919281950124925073115131277954891876532072848 9e+0000
 *       kc[3] =  -4.2502733997955757397602959692931920700944409036 6e-0001
 *       kc[4] =   7.4908066165099009610967295461831762388842162861 3e-0002
 *       kc[5] =  -8.2145886611128228798553946417397655543605021512 0e-0003
 *       kc[6] =   6.1420257880952922850320525516576120475021160340 2e-0004
 *       kc[7] =  -3.3307343269114960700721733030259526717954590874 0e-0005
 *       kc[8] =   1.3697095904783208579680974546153086559799368020 4e-0006
 *       kc[9] =  -4.4178077426258351445024651272720180621727109733 6e-0008
 *       kc[10]=   1.1474140921238185882001656766448812347866070575 9e-0009
 *       kc[11]=  -2.4426123611470737455843750065438100630050274963 2e-0011
 *
 *      Double precision, remez < 2**(61.91)
 *                                 2            4                  12
 *          kpcos(t) = 1/pi + pi/2 *t +  kc[2] * t  + ... + kc[6] * t
 *
 *       kc[2] =   1.2919281950123022495328358672257576618955196600 8e+0000
 *       kc[3] =  -4.2502733994014951850015885075339317351973214921 3e-0001
 *       kc[4] =   7.4908062518701531237392514221942942237555672775 2e-0002
 *       kc[5] =  -8.2144204090609921086697735228405484905134869271 5e-0003
 *       kc[6] =   6.1041135682951541457556656473363253233390411596 8e-0004
 *
 *      Single precision, remez < 2**(-30.13)
 *                                 2                  6
 *          kpcos(t) = kc[0] +  kc[1] * t  + ... + kc[3] * t
 *
 *       kc[0] =   3.1830988618379067153776752674502872406891929148 0e-0001
 *       kc[1] =  -1.5707958144776256819946787506585453862659493779 1e+0000
 *       kc[2] =   1.2918352809255869284407300402956867402780739386 2e+0000
 *       kc[3] =  -4.2023294977130768598101591442519547160273907553 7e-0001
 *
 *      Computation note: in simulating higher precision arithmetic, kcpcos
 *      return head = 1/pi chopped, and tail = pi/2 *t^2 + (tail part of 1/pi
 *      + ...) to maintain extra bits precision. In particular, pi/2 * t^2
 *      is calculated with great care.
 *
 *      Thus, the computation of gamma(-x), x>0, is:
```



```
 *      Let k = int(x), z = x-k.
 *      For z in (I)
 *                                      k+1
 *                                  (-1)
 *              gamma(-x) = ------------------- ;
 *                          kpsin(z)*gamma(1+x)
 *
 *      otherwise, for z in (II),
 *                                      k+1
 *                                  (-1)
 *              gamma(-x) = ----------------------- ;
 *                          kpcos(0.5-z)*gamma(1+x)
 *
 *      otherwise, for z in (III),
 *                                      k+1
 *                                  (-1)
 *              gamma(-x) = -------------------- .
 *                          kpsin(1-z)*gamma(1+x)
 *
 *      Thus, the computation of gamma(-x) reduced to the computation of
 *      gamma(1+x) and kpsin(), kpcos().
 *
 * (B) For x between 1 and 2.  We break [1,2] into three parts:
 *      GT1 = [1.0000, 1.2845]
 *      GT2 = [1.2844, 1.6374]
 *      GT3 = [1.6373, 2.0000]
 *
 *    For x in GTi, i=1,2,3, let
 *      z1  =  1.1348618057327907696897939357746529170 06
 *      gz1 = gamma(z1)  =   0.9382046279096824494097535615803269576988
 *      tz1 = gamma'(z1) =  -0.3517214357852935791015625000000000000000
 *
 *      z2  =  1.4616321449683623412626595423257213284 68e+0000
 *      gz2 = gamma(z2)  = 0.8856031944108887002788159005825887332080
 *      tz2 = gamma'(z2) = 0.00
 *
 *      z3  =  1.8197731011005006017878687049216069963 12e+0000
 *      gz3 = gamma(z3)  = 0.9367814114636523216188468970808378497426
 *      tz3 = gamma'(z3) = 0.2805306315422058105468750000000000000000
 *
 *    and
 *      y = x-zi        ... for extra precision, write y = y.h + y.l
 *    Then
 *      gamma(x) = gzi + tzi*(y.h+y.l) + y*y*Ri(y),
 *               = gzi.h + (tzi*y.h + ((tzi*y.l+gzi.l) +  y*y*Ri(y)))
 *               = gy.h + gy.l
 *    where
 *      (I) For double precision
 *
 *              Ri(y) = Pi(y)/Qi(y), i=1,2,3;
 *
 *              P1(y) = p1[0] + p1[1]*y + ... + p1[4]*y^4
 *              Q1(y) = q1[0] + q1[1]*y + ... + q1[5]*y^5
 *
 *              P2(y) = p2[0] + p2[1]*y + ... + p2[3]*y^3
 *              Q2(y) = q2[0] + q2[1]*y + ... + q2[6]*y^6
 *
 *              P3(y) = p3[0] + p3[1]*y + ... + p3[4]*y^4
 *              Q3(y) = q3[0] + q3[1]*y + ... + q3[5]*y^5
 *
 *              Remez precision of Ri(y):
 *              |gamma(x)-(gzi+tzi*y) - y*y*Ri(y)|  <= 2**-62.3 ... for i = 1
 *                                                  <= 2**-59.4 ... for i = 2
 *                                                  <= 2**-62.1 ... for i = 3
 *
 *      (II) For quad precision
```

```
 *
 *              Ri(y) = Pi(y)/Qi(y), i=1,2,3;
 *
 *              P1(y) = p1[0] + p1[1]*y + ... + p1[9]*y^9
 *              Q1(y) = q1[0] + q1[1]*y + ... + q1[8]*y^8
 *
 *              P2(y) = p2[0] + p2[1]*y + ... + p2[9]*y^9
 *              Q2(y) = q2[0] + q2[1]*y + ... + q2[9]*y^9
 *
 *              P3(y) = p3[0] + p3[1]*y + ... + p3[9]*y^9
 *              Q3(y) = q3[0] + q3[1]*y + ... + q3[9]*y^9
 *
 *              Remez precision of Ri(y):
 *              |gamma(x)-(gzi+tzi*y) - y*y*Ri(y)|  <= 2**-118.2 ... for i = 1
 *                                                  <= 2**-126.8 ... for i = 2
 *                                                  <= 2**-119.5 ... for i = 3
 *
 *      (III) For single precision
 *
 *              Ri(y) = Pi(y), i=1,2,3;
 *
 *              P1(y) = p1[0] + p1[1]*y + ... + p1[5]*y^5
 *
 *              P2(y) = p2[0] + p2[1]*y + ... + p2[5]*y^5
 *
 *              P3(y) = p3[0] + p3[1]*y + ... + p3[4]*y^4
 *
 *              Remez precision of Ri(y):
 *              |gamma(x)-(gzi+tzi*y) - y*y*Ri(y)|  <= 2**-30.8 ... for i = 1
 *                                                  <= 2**-31.6 ... for i = 2
 *                                                  <= 2**-29.5 ... for i = 3
 *
 *    Notes. (1) GTi and zi are choosen to balance the interval width and
 *               minimize the distant between gamma(x) and the tangent line at
 *               zi. In particular, we have
 *               |gamma(x)-(gzi+tzi*(x-zi))|  <=   0.01436... for x in [1,z2]
 *                                            <=   0.01265... for x in [z2,2]
 *
 *           (2) zi are slightly adjusted so that tzi=gamma'(zi) is very
 *               close to a single precision value.
 *
 *    Coefficents: Single precision
 *      i= 1:
 *       P1[0] =   7.0908725343508836027145161339801928007756127944 3e-0001
 *       P1[1] =  -5.1722956078865210854514197823870179010524176108 9e-0001
 *       P1[2] =   5.2340339452815078940582522223237706471623377643 27e-0001
 *       P1[3] =  -4.5458630871707501078404156606948041173263481489 9e-0001
 *       P1[4] =   4.2059649091523908545996459055925691349819095523 3e-0001
 *      P1[5] =  -3.5730758971237752097833218583824145864214218578 9e-0001
 *
 *      i = 2:
 *       p2[0] =   4.2848698398029519816605611922398428443426434457 8e-0001
 *       p2[1] =  -1.3070453948770913852868012162789973538665010391 4e-0001
 *       p2[2] =   1.6085628503805195507286121935265585154295543087 1e-0001
 *       p2[3] =  -9.2228516134601058377445880206737118215893794350 7e-0002
 *       p2[4] =   7.1924051176722526074089029260507059556062617935 7e-0002
 *       p2[5] =  -4.8815826559335509370311223853448463619326045957 4e-0002
 *
 *      i = 3
 *       p3[0] =   3.8240953111880775908112147978609213481480887288 0e-0001
 *       p3[1] =   2.6530988818018864795640040301349575936516785342 6e-0002
 *       p3[2] =   8.0681510977507917192356116941537030937629673983 5e-0002
 *       p3[3] =  -1.5482159166613761392884089083517435167400776479 9e-0002
 *       p3[4] =   1.7630823924271726853049831341689918815716518340 5e-0002
 *
 *    Coefficents: Double precision
```



```
 *      i = 1:
 *       p1[0]   =   0.7090868361997779700800492719281464815139770507812 5000
 *       p1[1]   =   1.7198706139304855808957951338435644166835172006 1e-0001
 *       p1[2]   =  -3.1927334579199097029332031612281396052770545067 1e-0002
 *       p1[3]   =   8.3617264541911003626716960039054997356353447698 9e-0003
 *       p1[4]   =   1.1374533664857283833315221347427797124462975810 1e-0003
 *       q1[0]   =   1.0
 *       q1[1]   =   9.7198021782603293752646073177847238979132196808 2e-0001
 *       q1[2]   =  -7.4357674332675617659408413725604265349708766603 0e-0002
 *       q1[3]   =  -1.1934594493226555976971947051510201224699525537 2e-0001
 *       q1[4]   =   1.5991344575142500262093512047078138221505028476 2e-0002
 *       q1[5]   =   1.1260113685337498456657269130640232191154755078 3e-0003
 *      i = 2:
 *       p2[0]   =   0.4284868158555860118141822567849885672330856323242 1875
 *       p2[1]   =   6.5359676266897081602371884510566741848312210362 9e-0002
 *       p2[2]   =  -6.9728082963121293132105077092512826427276893673 1e-0003
 *       p2[3]   =   6.4634235902198171894720860567481326016611663289 9e-0003
 *       q2[0]   =   1.0
 *       q2[1]   =   4.5757262056050604706255395745406201232751931393 6e-0001
 *       q2[2]   =  -2.5218259488607545285965500340779610308342257203 6e-0001
 *       q2[3]   =  -1.8297094540777859468134816604010319717871155282 7e-0002
 *       q2[4]   =   2.4357472699316956647522764212883014130495384050 2e-0002
 *       q2[5]   =  -5.2039040646694252535864595756489741125866708550 1e-0003
 *       q2[6]   =   4.7952025138327983763555243198802325603195113388 5e-0004
 *      i = 3:
 *       p3[0]   =   0.3824094797345674590083319799305172637104988098144531 25
 *       p3[1]   =   1.4287604869766816159906981404344930157292803414 0e-0001
 *       p3[2]   =   3.4215757105225053681792386601356176078574889907 1e-0003
 *       p3[3]   =  -5.0154262171006752140508788785699170098770927293 7e-0004
 *       p3[4]   =   8.8928581486674091012383468816383828761833212267 0e-0004
 *       q3[0]   =   1.0
 *       q3[1]   =   3.0425308662944420100221564094895789790629963316 8e-0001
 *       q3[2]   =  -2.2316240737999947728255567283488121387318552000 6e-0001
 *       q3[3]   =  -1.0506086774195206592180981193367013142755290363 6e-0002
 *       q3[4]   =   1.7051176391618698247330186198085635200592666932 0e-0002
 *       q3[5]   =  -2.1295020168360918792789941670009463076418247746 4e-0003
 *
 *    Note that all pi0 are exact in double, which is obtained by a
 *    special Remez Algorithm.
 *
 *    Coefficents: Quad precision
 *      i = 1:
 *       p1[0] =   0.7090868361997779190371857415076101246115137205 57
 *       p1[1] =   4.4575478120648903582791596936735483566739160695 1e-0001
 *       p1[2] =   3.2104929873583238231166227388263221006291815385 2e-0002
 *       p1[3] =  -5.7129679634210661765176524585828919736968886435 0e-0003
 *       p1[4] =   6.0466689289199897708161917496985583160696535277 3e-0003
 *       p1[5] =   8.9910618699688871193962781217476525882265864516 8e-0004
 *       p1[6] =  -6.9649684614440774143120700852701844181017556894 9e-0005
 *       p1[7] =   1.5259704611898402081422540930013144507021388242 9e-0005
 *       p1[8] =   5.6852107616849567384471146540743218919068154154 7e-0007
 *       p1[9] =   3.3074967351963489522058206252028656561041895297 9e-0008
 *       q1[0] =   1.0+0000
 *       q1[1] =   1.3580651172167107040857085353725707957949065066 8e+0000
 *       q1[2] =   2.9756781015342955340532714009606308699407295296 1e-0001
 *       q1[3] =  -1.5295683598258857150295437282168185168111809787 0e-0001
 *       q1[4] =  -2.8824851956142010976878161528908205359795452121 8e-0002
 *       q1[5] =   1.0347531171993740521978994845631393630237839595 5e-0002
 *       q1[6] =   4.1231020324389122368965360124391297374822742313 e-0004
 *       q1[7] =  -3.1265370815229086724893192512038072951833250738 8e-0004
 *       q1[8] =   2.3667217085040974523735810566775776052701433245 8e-0005
 *
 *      i = 2:
 *       p2[0] =   0.4284868158555854297302099078106506167377566974 77
 *       p2[1] =   2.6362212406788522291919265115158154194336261735 2e-0001
 *       p2[2] =   3.8552068367002886573187727674139042174497144685 5e-0002
```

```
 391  *       p2[3] =   3.0506597827812854995889713319029532525802352586 2e-0003
 392  *       p2[4] =   2.4823293495172312889208041505408433915245044508 1e-0003
 393  *       p2[5] =   3.6709277706563236069331376222141154774155010540 7e-0004
 394  *       p2[6] =   3.8122804561608578967453090256314525053219451894 6e-0006
 395  *       p2[7] =   4.6167722586708755405953145513383917582253761767 7e-0006
 396  *       p2[8] =   2.1820905238570320043823920099120191660936487299 3e-0007
 397  *       p2[9] =   1.0049053898524584646000624406562475442102254245 4e-0008
 398  *       q2[0] =   1.0
 399  *       q2[1] =   9.2027635020763929056778372527312854422457077505 6e-0001
 400  *       q2[2] =  -4.7953368365416510744802051573388378113894777149 5e-0003
 401  *       q2[3] =  -1.2453833758589930049444460024868790194768429168 3e-0001
 402  *       q2[4] =   4.4986605076347235854752470843171911420453549141 2e-0003
 403  *       q2[5] =   7.2071545569792056062163832535629264060407859190 7e-0003
 404  *       q2[6] =  -8.6851316902912678028079833709198278059822809611 6e-0004
 405  *       q2[7] =  -1.2510443162940118152502709822274554480997422987 4e-0004
 406  *       q2[8] =   3.1055834483900003848919130455099804752125343746 4e-0005
 407  *       q2[9] =  -1.7682922785285217601853713957360943365250676571 2e-0006
 408  *
 409  *      i = 3
 410  *       p3[0] =   0.38240947973456750485027476610753556400704393889 02
 411  *       p3[1] =   3.4219809307661849541585490633590842715983337777 4e-0001
 412  *       p3[2] =   9.6382189500585568303961406863153237440702754858 e-0002
 413  *       p3[3] =   8.7606942104269638485246204418852025215684676866 7e-0003
 414  *       p3[4] =   1.8647789038916149122487201414930901526189753748 8e-0003
 415  *       p3[5] =   8.1687135454030989587997474285370131154128694419 1e-0004
 416  *       p3[6] =   6.8378348367460032251869509086465938165012562521 6e-0005
 417  *       p3[7] =  -1.1016826971926157470856593517271920927219082845 6e-0006
 418  *       p3[8] =   9.6624322850838042015923485327890671706562972101 6e-0007
 419  *       p3[9] =   2.3185888557917725054116382067112166497433472814 2e-0008
 420  *       q3[0] =   1.0
 421  *       q3[1] =   8.2547982116881363463243743009037625251279306733 9e-0001
 422  *       q3[2] =  -1.6225136307393776973963962366929511034601557632 0e-0002
 423  *       q3[3] =  -1.1062128690591673275874513062942655969118757985 2e-0001
 424  *       q3[4] =   3.4830969397098561264444641578923001551536529145 9e-0003
 425  *       q3[5] =   6.7355373748748833303243126113128967234704340132 8e-0003
 426  *       q3[6] =  -7.6322200839337263016274358781100461305024512805 1e-0004
 427  *       q3[7] =  -1.3579267066919063147678476896195371177307325133 6e-0004
 428  *       q3[8] =   3.1961015095422358700622073006560815646020569061 8e-0005
 429  *       q3[9] =  -1.8209655386282234661010952201512958569335434832 2e-0006
 430  *
 431  * (C) For x between 0 and 1.
 432  *     Let P stand for the number of significant bits in the working precision.
 433  *                          -P                                  1
 434  *    (1)For 0 <= x <= 2   , gamma(x) is computed by --- rounded to nearest.
 435  *                                                              x
 436  *       The error is bound by 0.739 ulp(gamma(x)) in IEEE double precision.
 437  *      Proof.
 438  *                 1                        2
 439  *      Since  --------  ~  x + 0.577...*x  - ...,  we have, for small x,
 440  *               gamma(x)
 441  *           1                    1
 442  *      ----------- < gamma(x) < --- and
 443  *      x(1+0.578x)               x
 444  *               1                1           1
 445  *        0 <  --- - gamma(x) <= ---  -  ----------- < 0.578
 446  *               x                x      x(1+0.578x)
 447  *                                     1        1                        -P
 448  *      The error is thus bounded by --- ulp(---) + 0.578. Since x <= 2   ,
 449  *                                     2        x
 450  *       1      P       1           P                                     1
 451  *      --- >= 2 , ulp(---) >= ulp(2  ) >= 2. Thus 0.578=0.289*2<=0.289ulp(-)
 452  *       x             x                                                  x
 453  *       Thus
 454  *                                 1                                   1
 455  *               | gamma(x) - [---] rounded | <= (0.5+0.289)*ulp(---).
 456  *                                 x                                   x
```



```
 457  *                          -P                                  1
 458  *      Note that for x<= 2  , it is easy to see that ulp(---)=ulp(gamma(x))
 459  *                                                              x
 460  *                             n                                1
 461  *      except only when x = 2 , (n<= -53). In such cases, --- is exact
 462  *                                                              x
 463  *      and therefore the error is bounded by
 464  *                          1
 465  *              0.298*ulp(---) = 0.298*2*ulp(gamma(x)) = 0.578ulp(gamma(x)).
 466  *                          x
 467  *      Thus we conclude that the error in gamma is less than 0.739 ulp.
 468  *
 469  *    (2)Otherwise, for x in GTi-1 (see B), let y = x-(zi-1). From (B) we obtain
 470  *                                                              gamma(1+x)
 471  *      gamma(1+x) = gy.h + gy.l,  then compute gamma(x) by -----------.
 472  *                                                                   x
 473  *                                                              gy.h
 474  *      Implementaion note. Write x = x.h+x.l, and Let th = ----- chopped to
 475  *                                                                x
 476  *      20 bits, then
 477  *                                  gy.h+gy.l
 478  *              gamma(x) = th + (----------  - th )
 479  *                                       x
 480  *                                   1
 481  *                        = th + ---*(gy.h-th*x.h+gy.l-th*x.l)
 482  *                                   x
 483  *
 484  * (D) For x between 2 and 8. Let n = 1+x chopped to an integer. Then
 485  *
 486  *               gamma(x)=(x-1)*(x-2)*...*(x-n)*gamma(x-n)
 487  *
 488  *     Since x-n is between 1 and 2, we can apply (B) to compute gamma(x).
 489  *
 490  *     Implementation detail. The computation of (x-1)(x-2)...(x-n) in simulated
 491  *     higher precision arithmetic can be somewhat optimized.  For example, in
 492  *     computing (x-1)*(x-2)*(x-3)*(x-4), if we compute (x-1)*(x-4) = z.h+z.l,
 493  *     then (x-2)(x-3) = z.h+2+z.l readily. In below, we list the expression
 494  *     of the formula to compute gamma(x).
 495  *
 496  *     Assume x-n is in GTi (i=1,2, or 3, see B for detail). Let y = x - n - zi.
 497  *     By (B) we have gamma(x-n) = gy.h+gy.l. If x = x.h+x.l, then we have
 498  *      n=1 (x in [2,3]):
 499  *        gamma(x) = (x-1)*gamma(x-1) = (x-1)*(gy.h+gy.l)
 500  *                 = [(x.h-1)+x.l]*(gy.h+gy.l)
 501  *      n=2 (x in [3,4]):
 502  *        gamma(x) = (x-1)(x-2)*gamma(x-2) = (x-1)*(x-2)*(gy.h+gy.l)
 503  *                 = ((x.h-2)+x.l)*((x.h-1)+x.l)*(gy.h+gy.l)
 504  *                 = [x.h*(x.h-3)+2+x.l*(x+(x.h-3))]*(gy.h+gy.l)
 505  *      n=3 (x in [4,5])
 506  *        gamma(x) = (x-1)(x-2)(x-3)*(gy.h+gy.l)
 507  *                 = (x.h*(x.h-3)+2+x.l*(x+(x.h-3)))*[((x.h-3)+x.l)(gy.h+gy.l)]
 508  *      n=4 (x in [5,6])
 509  *        gamma(x) = [(x-1)(x-4)]*[(x-2)(x-3)]*(gy.h+gy.l)
 510  *                 = [(x.h*(x.h-5)+4+x.l(x+(x.h-5)))]*[(x-2)*(x-3)]*(gy.h+gy.l)
 511  *                 = (y.h+y.l)*(y.h+1+y.l)*(gy.h+gy.l)
 512  *      n=5 (x in [6,7])
 513  *        gamma(x) = [(x-1)(x-4)]*[(x-2)(x-3)]*[(x-5)*(gy.h+gy.l)]
 514  *      n=6 (x in [7,8])
 515  *        gamma(x) = [(x-1)(x-6)]*[(x-2)(x-5)]*[(x-3)(x-4)]*(gy.h+gy.l)]
 516  *                 = [(y.h+y.l)(y.h+4+y.l)][(y.h+6+y.l)(gy.h+gy.l)]
 517  *
 518  * (E)Overflow Thresold. For x > Overflow thresold of gamma,
 519  *    return huge*huge (overflow).
 520  *
 521  *    By checking whether lgamma(x) >= 2**{128,1024,16384}, one can
 522  *    determine the overflow threshold for x in single, double, and
```

```
 *    quad precision. See over.c for details.
 *
 *    The overflow threshold of gamma(x) are
 *
 *    single: x = 3.5040096283e+01
 *              = 0x420C290F (IEEE single)
 *    double: x = 1.7162437695630271150 5e+02
 *              = 0x406573FAE561F647 (IEEE double)
 *    quad:   x = 1.7555483429044629170038892160702032034177e+03
 *              = 0x4009B6E3180CD66A5C4206F128BA77F4  (quad)
 *
 * (F)For overflow_threshold >= x >= 8, we use asymptotic approximation.
 *    (1) Stirling's formula
 *
 *      log(G(x)) ~= (x-.5)*(log(x)-1) + .5(log(2*pi)-1) + (1/x)*P(1/(x*x))
 *                 = L1 + L2 + L3,
 *    where
 *              L1(x) = (x-.5)*(log(x)-1),
 *              L2    = .5(log(2pi)-1) = 0.41893853....,
 *              L3(x) = (1/x)P(1/(x*x)),
 *
 *    The range of L1,L2, and L3 are as follows:
 *
 *      ------------------------------------------------------------------
 *      Range(L1) =  (single) [8.09..,88.30..]   =[2** 3.01..,2**  6.46..]
 *                   (double) [8.09..,709.3..]   =[2** 3.01..,2**  9.47..]
 *                   (quad)   [8.09..,11356.10..]=[2** 3.01..,2** 13.47..]
 *      Range(L2) = 0.41893853.....
 *      Range(L3) = [0.0104...., 0.00048....]    =[2**-6.58..,2**-11.02..]
 *      ------------------------------------------------------------------
 *
 *    Gamma(x) is then computed by exp(L1+L2+L3).
 *
 *    (2) Error analysis of (F):
 *    --------------------------
 *    The error in Gamma(x) depends on the error inherited in the computation
 *    of L= L1+L2+L3. Let L' be the computed value of L. The absolute error
 *    in L' is t = L-L'. Since exp(L') = exp(L-t) = exp(L)*exp(t) ~
 *    (1+t)*exp(L), the relative error in exp(L') is approximately t.
 *
 *    To guarantee the relatively accuracy in exp(L'), we would like
 *    |t| < 2**(-P-5) where P denotes for the number of significant bits
 *    of the working precision. Consequently, each of the L1,L2, and L3
 *    must be computed with absolute error bounded by 2**(-P-5) in absolute
 *    value.
 *
 *    Since L2 is a constant, it can be pre-computed to the desired accuracy.
 *    Also |L3| < 2**-6; therefore, it suffices to compute L3 with the
 *    working precision.  That is,
 *      L3(x) approxmiate log(G(x))-(x-.5)(log(x)-1)-.5(log(2pi)-1)
 *    to a precision bounded by 2**(-P-5).
 *
 *                                  2**(-6)
 *                            _________V_________________
 *              L1(x):     |_________|___________________|
 *                          __ _______________________
 *              L2:               |__|__________________________|
 *                                   _____________________________
 *         +    L3(x):               |____________________________|
 *                       -----------------------------------------
 *                         [leading] + [Trailing]
 *
 *    For L1(x)=(x-0.5)*(log(x)-1), we need ilogb(L1(x))+5 extra bits for
 *    both multiplicants to guarantee L1(x)'s absolute error is bounded by
 *    2**(-P-5) in absolute value. Here ilogb(y) is defined to be the unbias
 *    binary exponent of y in IEEE format.  We can get x-0.5 to the desire
```



```
 *    accuracy easily. It remains to compute log(x)-1 with ilogb(L1(x))+5
 *    extra bits accracy. Note that the range of L1 is 88.30.., 709.3.., and
 *    11356.10... for single, double, and quadruple precision, we have
 *
 *                            single     double      quadruple
 *                         -------------------------------------
 *      ilogb(L1(x))+5 <=     11          14           18
 *                         -------------------------------------
 *
 *    (3) Table Driven Method for log(x)-1:
 *    -------------------------------------
 *    Let x = 2**n * y, where 1 <= y < 2. Let Z={z(i),i=1,...,m}
 *    be a set of predetermined evenly distributed floating point numbers
 *    in [1, 2]. Let z(j) be the closest one to y, then
 *      log(x)-1 = n*log(2)-1  +  log(y)
 *               = n*log(2)-1  +  log(z(j)*y/z(j))
 *               = n*log(2)-1  +  log(z(j))  +  log(y/z(j))
 *               = T1(n)       +  T2(j)      +  T3,
 *
 *    where T1(n) = n*log(2)-1 and T2(j) = log(z(j)). Both T1 and T2 can be
 *    pre-calculated and be looked-up in a table. Note that 8 <= x < 1756
 *    implies 3<=n<=10 implies 1.079.. < T1(n) < 6.931.
 *
 *                        y-z(i)          y       1+s
 *    For T3, let s = --------; then ----- =  ----- and
 *                        y+z(i)         z(i)     1-s
 *                 1+s           2   3    2   5
 *      T3 = log(-----) = 2s + --- s  + --- s  + ....
 *                 1-s           3        5
 *
 *    Suppose the first term 2s is compute in extra precision. The
 *    dominating error in T3 would then be the rounding error of the
 *    second term 2/3*s**3. To force the rounding bounded by
 *    the required accuracy, we have
 *        single:  |2/3*s**3| < 2**-11   == > |s|<0.09014...
 *        double:  |2/3*s**3| < 2**-14   == > |s|<0.04507...
 *        quad  :  |2/3*s**3| < 2**-18   == > |s|<0.01788... = 2**(-5.80..)
 *
 *    Base on this analysis, we choose Z = {z(i)|z(i)=1+i/64+1/128, 0<=i<=63}.
 *    For any y in [1,2), let j = [64*y] chopped to integer, then z(j) is
 *    the closest to y, and it is not difficult to see that |s| < 2**(-8).
 *    Please note that the polynomial approximation of T3 must be accurate
 *        -24-11   -35    -53-14    -67          -113-18   -131
 *    to 2       =2    ,  2       = 2    ,  and  2        =2
 *    for single, double, and quadruple precision respectively.
 *
 *    Inplementation notes.
 *    (1) Table look-up entries for T1(n) and T2(j), as well as the calculation
 *        of the leading term 2s in T3,  are broken up into leading and trailing
 *        part such that (leading part)* 2**24 will always be an integer. That
 *        will guarantee the addition of the leading parts will be exact.
 *
 *                                  2**(-24)
 *                            _________V_________________
 *              T1(n):     |_________|___________________|
 *                          _______ _____________________
 *              T2(j):       |_______|______________________|
 *                           ____ _________________________
 *              2s:            |____|__________________________|
 *                                  ____________________________
 *         +    T3(s)-2s:           |____________________________|
 *                       -----------------------------------------
 *                         [leading] + [Trailing]
 *
 *    (2) How to compute 2s accurately.
```

```
 *        (A) Compute v = 2s to the working precision. If |v| < 2**(-18),
 *            stop.
 *        (B) chopped v to 2**(-24): v = ((int)(v*2**24))/2**24
 *       (C) 2s = v + (2s - v), where
 *                          1
 *              2s - v = --- * (2(y-z) - v*(y+z) )
 *                        y+z
 *                          1
 *                      = --- * ( [2(y-z) - v*(y+z)_h ]  - v*(y+z)_l  )
 *                        y+z
 *            where (y+z)_h = (y+z) rounded to 24 bits by (double)(float),
 *          and (y+z)_l = ((z+z)-(y+z)_h)+(y-z).  Note the the quantity
 *          in [] is exact.
 *                                                          2         4
 *    (3) Remez approximation for (T3(s)-2s)/s = T3[0]*s + T3[1]*s + ...:
 *       Single precision: 1 term (compute in double precision arithmetic)
 *          T3(s) = 2s + S1*s^3, S1 = 0.66667172318485180546936236975392 30
 *          Remez error: |T3(s)/s - (2s+S1*s^3)| < 2**(-35.87)
 *       Double precision: 3 terms, Remez error is bounded by 2**(-72.40),
 *          see "tgamma_log"
 *       Quad precision: 7 terms, Remez error is bounded by 2**(-136.54),
 *          see "tgammal_log"
 *
 *   The computation of 0.5*(ln(2pi)-1):
 *      0.5*(ln(2pi)-1) =  0.41893853320467274178032973640561763986 14...
 *      split 0.5*(ln(2pi)-1) to hln2pi_h + hln2pi_l, where hln2pi_h is the
 *      leading 21 bits of the constant.
 *          hln2pi_h= 0.41893839836120605468 75
 *          hln2pi_l= 1.34843466687092829736405617639861217364 8e-07
 *
 *   The computation of 1/x*P(1/x^2) = log(G(x))-(x-.5)(ln(x)-1)-(.5ln(2pi)-1):
 *      Let s = 1/x <= 1/8 < 0.125. We have
 *      quad precision
 *          |GP(s) - s*P(s^2)| <= 2**(-120.6), where
 *                                 3      5            39
 *          GP(s) = GP0*s+GP1*s +GP2*s +... +GP19*s    ,
 *       GP0  =   0.08333333333333333333333333333333172839171301
 *                      hex 0x3ffe5555 55555555 55555555 55555548
 *       GP1  =  -2.77777777777777777777777777492501211999399424104e-0003
 *       GP2  =   7.93650793650793650793635650541638236350020883243e-0004
 *       GP3  =  -5.95238095238095238057299772679324503339241961704e-0004
 *       GP4  =   8.41750841750841696138422987977683524926142600321e-0004
 *       GP5  =  -1.91752691752686682825032547823699662178842123308e-0003
 *       GP6  =   6.41025641022403480921891559356473451161279359322e-0003
 *       GP7  =  -2.95506535798414019189819587455577003732808185071e-0002
 *       GP8  =   1.79644367229970031486079180060923073476568732136e-0001
 *       GP9  =  -1.39243086487274662174562872567057200255649290646e+0000
 *       GP10 =   1.34025874044417962188677816477842265259608269775e+0001
 *       GP11 =  -1.56803713480127469414495545399982508700748274318e+0002
 *       GP12 =   2.18739841656201561694927630335099313968924493891e+0003
 *       GP13 =  -3.55249848644100338419187038090925410976237921269e+0004
 *       GP14 =   6.43464880437835286216768959439484376449179576452e+0005
 *       GP15 =  -1.20459154385577014992600342782821389605893904624e+0007
 *       GP16 =   2.09263249637351298563934942349749718491071093210e+0008
 *       GP17 =  -2.96247483183169219343745316433899599834685703457e+0009
 *       GP18 =   2.88984933605896033154727626086506756972327292981e+0010
 *       GP19 =  -1.40960434146030007732838382416230610302678063984e+0011
 *
 *       double precision
 *          |GP(s) - s*P(s^2)| <= 2**(-63.5), where
 *                                 3      5      7      9      11      13      15
 *          GP(s) = GP0*s+GP1*s +GP2*s +GP3*s +GP4*s +GP5*s  +GP6*s  +GP7*s  ,
 *
 *              GP0=  0.08333333333333332870740640640618477 (3FB55555 55555555)
 *              GP1= -2.77777777776649355200565611114627670089130772843e-0003
 *              GP2=  7.93650787486083724805476194170211775784158551509e-0004
```



```
 *              GP3= -5.95236628558314928757811419580281294593903582971e-0004
 *              GP4=  8.41566473999853451983137162780427812781178932540e-0004
 *              GP5= -1.90424776670441373564512942038926168175921303212e-0003
 *              GP6=  5.84933161530949666312333949534482303007354299178e-0003
 *              GP7= -1.59453228931082030262124832506144392496561694550e-0002
 *       single precision
 *          |GP(s) - s*P(s^2)| <= 2**(-37.78), where
 *                                 3      5
 *          GP(s) = GP0*s+GP1*s +GP2*s
 *        GP0 =   8.33333330959694065245736888749042811909994573178e-0002
 *        GP1 =  -2.77765545601667179767706600890361535225507762168e-0003
 *        GP2 =   7.77830853479775281781085278324621033523037489883e-0004
 *
 *
 *      Implementation note:
 *      z = (1/x), z2 = z*z, z4 = z2*z2;
 *      p = z*(GP0+z2*(GP1+....+z2*GP7))
 *        = z*(GP0+(z4*(GP2+z4*(GP4+z4*GP6))+z2*(GP1+z4*(GP3+z4*(GP5+z4*GP7)))))
 *
 *   Adding everything up:
 *      t = rr.h*ww.h+hln2pi_h                  ... exact
 *      w = (hln2pi_l + ((x-0.5)*ww.l+rr.l*ww.h)) + p
 *
 *   Computing exp(t+w):
 *      s = t+w; write s = (n+j/32)*ln2+r, |r|<=(1/64)*ln2, then
 *      exp(s) = 2**n * (2**(j/32) + 2**(j/32)*expm1(r)), where
 *      expm1(r) = r + Et1*r^2 + Et2*r^3 + ... + Et5*r^6, and
 *      2**(j/32) is obtained by table look-up S[j]+S_trail[j].
 *      Remez error bound:
 *      |exp(r) - (1+r+Et1*r^2+...+Et5*r^6)| <= 2^(-63).
 */
/* END CSTYLED */
#endif /* ! codereview */

#include "libm.h"

#define __HI(x)         ((int *)&x)[HIWORD]
#define __LO(x)         ((unsigned *)&x)[LOWORD]
#define __HI(x) ((int *) &x)[HIWORD]
#define __LO(x) ((unsigned *) &x)[LOWORD]

struct Double {
        double h;
        double l;
};
_____unchanged_portion_omitted_

#define P10             cr[0]
#define P11             cr[1]
#define P12             cr[2]
#define P13             cr[3]
#define P14             cr[4]
#define Q10             cr[5]
#define Q11             cr[6]
#define Q12             cr[7]
#define Q13             cr[8]
#define Q14             cr[9]
#define Q15             cr[10]
#define P20             cr[11]
#define P21             cr[12]
#define P22             cr[13]
#define P23             cr[14]
#define Q20             cr[15]
#define Q21             cr[16]
#define Q22             cr[17]
#define Q23             cr[18]
```

```
1125 #define Q24             cr[19]
1126 #define Q25             cr[20]
1127 #define Q26             cr[21]
1128 #define P30             cr[22]
1129 #define P31             cr[23]
1130 #define P32             cr[24]
1131 #define P33             cr[25]
1132 #define P34             cr[26]
1133 #define Q30             cr[27]
1134 #define Q31             cr[28]
1135 #define Q32             cr[29]
1136 #define Q33             cr[30]
1137 #define Q34             cr[31]
1138 #define Q35             cr[32]

1140 static const double GZ1_h = +0.938204627909682398190,
1126 static const double
1127         GZ1_h = +0.938204627909682398190,
1141         GZ1_l = +5.121952600248205157935e-17,
1142         GZ2_h = +0.885603194410888749921,
1143         GZ2_l = -4.964236872556339810692e-17,
1144         GZ3_h = +0.936781411463652347038,
1145         GZ3_l = -2.541923110834479415023e-17,
1146         TZ1 = -0.351721435785293579101562 5,
1147         TZ3 = +0.280530631542205810546875;
1135 /* INDENT ON */

1149 /*
1150  * compute gamma(y=yh+yl) for y in GT1 = [1.0000, 1.2845]
1151  * assume yh got 20 significant bits
1152  */
1137 /* compute gamma(y=yh+yl) for y in GT1 = [1.0000, 1.2845] */
1138 /* assume yh got 20 significant bits */
1153 static struct Double
1154 GT1(double yh, double yl)
1155 {
1140 GT1(double yh, double yl) {
1156         double t3, t4, y, z;
1157         struct Double r;

1159         y = yh + yl;
1160         z = y * y;
1161         t3 = (z * (P10 + y * ((P11 + y * P12) + z * (P13 + y * P14)))) / (Q10 +
1162             y * ((Q11 + y * Q12) + z * ((Q13 + Q14 * y) + z * Q15)));
1146         t3 = (z * (P10 + y * ((P11 + y * P12) + z * (P13 + y * P14)))) /
1147                 (Q10 + y * ((Q11 + y * Q12) + z * ((Q13 + Q14 * y) + z * Q15)));
1163         t3 += (TZ1 * yl + GZ1_l);
1164         t4 = TZ1 * yh;
1165         r.h = (double)((float)(t4 + GZ1_h + t3));
1150         r.h = (double) ((float) (t4 + GZ1_h + t3));
1166         t3 += (t4 - (r.h - GZ1_h));
1167         r.l = t3;
1168         return (r);
1169 }

1171 /*
1172  * compute gamma(y=yh+yl) for y in GT2 = [1.2844, 1.6374]
1173  * assume yh got 20 significant bits
1174  */
1156 /* compute gamma(y=yh+yl) for y in GT2 = [1.2844, 1.6374] */
1157 /* assume yh got 20 significant bits */
1175 static struct Double
1176 GT2(double yh, double yl)
1177 {
1159 GT2(double yh, double yl) {
1178         double t3, y, z;
```



```
1179         struct Double r;

1181         y = yh + yl;
1182         z = y * y;
1183         t3 = (z * (P20 + y * P21 + z * (P22 + y * P23))) / (Q20 + (y * ((Q21 +
1184             Q22 * y) + z * Q23) + (z * z) * ((Q24 + Q25 * y) + z * Q26))) +
1185             GZ2_l;
1186         r.h = (double)((float)(GZ2_h + t3));
1165         t3 = (z * (P20 + y * P21 + z * (P22 + y * P23))) /
1166                 (Q20 + (y * ((Q21 + Q22 * y) + z * Q23) +
1167                 (z * z) * ((Q24 + Q25 * y) + z * Q26))) + GZ2_l;
1168         r.h = (double) ((float) (GZ2_h + t3));
1187         r.l = t3 - (r.h - GZ2_h);
1188         return (r);
1189 }

1191 /*
1192  * compute gamma(y=yh+yl) for y in GT3 = [1.6373, 2.0000]
1193  * assume yh got 20 significant bits
1194  */
1173 /* compute gamma(y=yh+yl) for y in GT3 = [1.6373, 2.0000] */
1174 /* assume yh got 20 significant bits */
1195 static struct Double
1196 GT3(double yh, double yl)
1197 {
1176 GT3(double yh, double yl) {
1198         double t3, t4, y, z;
1199         struct Double r;

1201         y = yh + yl;
1202         z = y * y;
1203         t3 = (z * (P30 + y * ((P31 + y * P32) + z * (P33 + y * P34)))) / (Q30 +
1204             y * ((Q31 + y * Q32) + z * ((Q33 + Q34 * y) + z * Q35)));
1182         t3 = (z * (P30 + y * ((P31 + y * P32) + z * (P33 + y * P34)))) /
1183                 (Q30 + y * ((Q31 + y * Q32) + z * ((Q33 + Q34 * y) + z * Q35)));
1205         t3 += (TZ3 * yl + GZ3_l);
1206         t4 = TZ3 * yh;
1207         r.h = (double)((float)(t4 + GZ3_h + t3));
1186         r.h = (double) ((float) (t4 + GZ3_h + t3));
1208         t3 += (t4 - (r.h - GZ3_h));
1209         r.l = t3;
1210         return (r);
1211 }

1192 /* INDENT OFF */
1214 /*
1215  * return tgamma(x) scaled by 2**-m for 8<x<=171.62... using Stirling's formula
1216  *     log(G(x)) ~= (x-.5)*(log(x)-1) + .5(log(2*pi)-1) + (1/x)*P(1/(x*x))
1217  *                  = L1 + L2 + L3,
1218  */
1198 /* INDENT ON */
1219 static struct Double
1220 large_gam(double x, int *m)
1221 {
1222         double z, t1, t2, t3, z2, t5, w, y, u, r, z4, v, t24 = 16777216.0, p24 =
1223             1.0 / 16777216.0;
1200 large_gam(double x, int *m) {
1201         double z, t1, t2, t3, z2, t5, w, y, u, r, z4, v, t24 = 16777216.0,
1202                 p24 = 1.0 / 16777216.0;
1224         int n2, j2, k, ix, j;
1225         unsigned lx;
1226         struct Double zz;
1227         double u2, ss_h, ss_l, r_h, w_h, w_l, t4;
```

```
/* INDENT OFF */
/*
 * compute ss = ss.h+ss.l = log(x)-1 (see tgamma_log.h for details)
 *
 *  log(x) - 1 = T1(n) + T2(j) + T3(s), where x = 2**n * y,  1<=y<2,
 *  j=[64*y], z[j]=1+j/64+1/128, s = (y-z[j])/(y+z[j]), and
 *       T1(n) = T1[2n,2n+1] = n*log(2)-1,
 *       T2(j) = T2[2j,2j+1] = log(z[j]),
 *       T3(s) = 2s + A1[0]s^3 + A2[1]s^5 + A3[2]s^7
 *  Note
 *  (1) the leading entries are truncated to 24 binary point.
 *  (2) Remez error for T3(s) is bounded by 2**(-72.4)
 *                                  2**(-24)
 *                              _________V_________________
 *              T1(n):     |_________|___________________|
 *                           _______ _____________________
 *              T2(j):       |_______|_____________________|
 *                             ____ _______________________
 *              2s:            |____|_______________________|
 *                                  _________________________
 *          +   T3(s)-2s:           |_________________________|
 *                       ----------------------------------------
 *                          [leading] + [Trailing]
 */
/* INDENT ON */
        ix = __HI(x);
        lx = __LO(x);
        n2 = (ix >> 20) - 0x3ff;                /* exponent of x, range:3-7 */
        n2 += n2;                               /* 2n */
        ix = (ix & 0x000fffff) | 0x3ff00000;    /* y = scale x to [1,2] */
        __HI(y) = ix;
        __LO(y) = lx;
        __HI(z) = (ix & 0xffffc000) | 0x2000;   /* z[j]=1+j/64+1/128 */
        __LO(z) = 0;
        j2 = (ix >> 13) & 0x7e;                 /* 2j */
        t1 = y + z;
        t2 = y - z;
        r = one / t1;
        t1 = (double)((float)t1);
        t1 = (double) ((float) t1);
        u = r * t2;                     /* u = (y-z)/(y+z) */
        t4 = T2[j2 + 1] + T1[n2 + 1];
        z2 = u * u;
        k = __HI(u) & 0x7fffffff;
        t3 = T2[j2] + T1[n2];

#endif /* ! codereview */
        if ((k >> 20) < 0x3ec) {        /* |u|<2**-19 */
                t2 = t4 + u * ((two + z2 * A1) + (z2 * z2) * (A2 + z2 * A3));
        } else {
                t5 = t4 + u * (z2 * A1 + (z2 * z2) * (A2 + z2 * A3));
                u2 = u + u;
                v = (double)((int)(u2 * t24)) * p24;
                v = (double) ((int) (u2 * t24)) * p24;
                t2 = t5 + r * ((two * t2 - v * t1) - v * (y - (t1 - z)));
                t3 += v;
        }

        ss_h = (double)((float)(t2 + t3));
        ss_h = (double) ((float) (t2 + t3));
        ss_l = t2 - (ss_h - t3);

        /*
         * compute ww = (x-.5)*(log(x)-1) + .5*(log(2pi)-1) + 1/x*(P(1/x^2)))
         * where ss = log(x) - 1 in already in extra precision
         */
```



```
        z = one / x;
        r = x - half;
        r_h = (double)((float)r);
        r_h = (double) ((float) r);
        w_h = r_h * ss_h + hln2pi_h;
        z2 = z * z;
        w = (r - r_h) * ss_h + r * ss_l;
        z4 = z2 * z2;
        t1 = z2 * (GP1 + z4 * (GP3 + z4 * (GP5 + z4 * GP7)));
        t2 = z4 * (GP2 + z4 * (GP4 + z4 * GP6));
        t1 += t2;
        w += hln2pi_l;
        w_l = z * (GP0 + t1) + w;
        k = (int)((w_h + w_l) * invln2_32 + half);
        k = (int) ((w_h + w_l) * invln2_32 + half);

        /* compute the exponential of w_h+w_l */
        j = k & 0x1f;
        *m = (k >> 5);
        t3 = (double)k;
        t3 = (double) k;

        /* perform w - k*ln2_32 (represent as w_h - w_l) */
        t1 = w_h - t3 * ln2_32hi;
        t2 = t3 * ln2_32lo;
        w = w_l - t2;
        w_h = t1 + w_l;
        w_l = t2 - (w_l - (w_h - t1));

        /* compute exp(w_h+w_l) */
        z = w_h - w_l;
        z2 = z * z;
        t1 = z2 * (Et1 + z2 * (Et3 + z2 * Et5));
        t2 = z2 * (Et2 + z2 * Et4);
        t3 = w_h - (w_l - (t1 + z * t2));
        zz.l = S_trail[j] * (one + t3) + S[j] * t3;
        zz.h = S[j];
        return (zz);
}


/* INDENT OFF */
/*
 * kpsin(x)= sin(pi*x)/pi
 *                3        5        7        9        11        13        15
 *       = x+ks[0]*x +ks[1]*x +ks[2]*x +ks[3]*x +ks[4]*x  +ks[5]*x  +ks[6]*x
 */
static const double ks[] = {
        -1.6449340668482264060656 9,
        +8.1174242528334165588366874187400892085069859062 1e-0001,
        -1.9075182412086287382559727911830494399404225829 1e-0001,
        +2.6147847763255427831728962833265453935352191157 0e-0002,
        -2.3460797851020271037761719027873552535434770586 6e-0003,
        +1.4841329229005169589724289997712184676382422170 5e-0004,
        -6.8773076963754348810868872677768726248535707224 2e-0006,
};
/* INDENT ON */

/* assume x is not tiny and positive */
static struct Double
kpsin(double x)
{
kpsin(double x) {
        double z, t1, t2, t3, t4;
        struct Double xx;
```

```
        z = x * x;
        xx.h = x;
        t1 = z * x;
        t2 = z * z;
        t4 = t1 * ks[0];
        t3 = (t1 * z) * ((ks[1] + z * ks[2] + t2 * ks[3]) + (z * t2) * (ks[4] +
            z * ks[5] + t2 * ks[6]));
        t3 = (t1 * z) * ((ks[1] + z * ks[2] + t2 * ks[3]) + (z * t2) *
                (ks[4] + z * ks[5] + t2 * ks[6]));
        xx.l = t4 + t3;
        return (xx);
}
```

```
/* INDENT OFF */
/*
 * kpcos(x)= cos(pi*x)/pi
 *                2        4        6        8        10        12
 *      = 1/pi +kc[0]*x +kc[1]*x +kc[2]*x +kc[3]*x +kc[4]*x  +kc[5]*x
 */

static const double one_pi_h = 0.31830988618379063570529 2970,
        one_pi_l = 3.583247455607534006714276420e-17;
static const double npi_2_h = -1.5625,
        npi_2_l = -0.008296326794896619231321691639750550995558832 23;
static const double kc[] = {
        -1.57079632679489661923132169163975055099555883223e+0000,
        +1.29192819501230224953283586722575766189551966008e+0000,
        -4.25027339940149518500158850753393173519732149213e-0001,
        +7.49080625187015312373925142219429422375556727752e-0002,
        -8.21442040906099210866977352284054849051348692715e-0003,
        +6.10411356829515414575566564733632532333904115968e-0004,
};
/* INDENT ON */
```

```
/* assume x is not tiny and positive */
static struct Double
kpcos(double x)
{
kpcos(double x) {
        double z, t1, t2, t3, t4, x4, x8;
        struct Double xx;

        z = x * x;
        xx.h = one_pi_h;
        t1 = (double)((float)x);
        t1 = (double) ((float) x);
        x4 = z * z;
        t2 = npi_2_l * z + npi_2_h * (x + t1) * (x - t1);
        t3 = one_pi_l + x4 * ((kc[1] + z * kc[2]) + x4 * (kc[3] + z * kc[4] +
            x4 * kc[5]));
        t3 = one_pi_l + x4 * ((kc[1] + z * kc[2]) + x4 * (kc[3] + z *
                kc[4] + x4 * kc[5]));
        t4 = t1 * t1;           /* 48 bits mantissa */
        x8 = t2 + t3;
        t4 *= npi_2_h;    /* npi_2_h is 5 bits const. The product is exact */
        xx.l = x8 + t4;   /* that will minimized the rounding error in xx.l */
        return (xx);
}
```

```
/* INDENT OFF */
static const double
/* 0.13486180573279076968979393577465291 7006 */
        /* 0.13486180573279076968979393577465291 7006 */
        t0z1 = 0.13486180573279077377 08,
        t0z1_l = -4.08100777085782990225 31e-18,
```



```
/* 0.46163214496836234126265954232572132846 8 */
        /* 0.46163214496836234126265954232572132846 8 */
        t0z2 = 0.46163214496836235678 50,
        t0z2_l = -1.55223481628586768905 21e-17,
/* 0.81977310110050060178786870492160699631 2 */
        /* 0.81977310110050060178786870492160699631 2 */
        t0z3 = 0.81977310110050061187 08,
        t0z3_l = -1.00829451224871034983 25e-17;

/*
 * 1.13486180573279076968979393577465291 7006
 */
        /* 1.13486180573279076968979393577465291 7006 */
/* INDENT ON */
```

```
/* gamma(x+i) for 0 <= x < 1  */
static struct Double
gam_n(int i, double x)
{
        struct Double rr = { 0.0L, 0.0L }, yy;
gam_n(int i, double x) {
        struct Double rr = {0.0L, 0.0L}, yy;
        double r1, r2, t2, z, xh, xl, yh, yl, zh, z1, z2, zl, x5, wh, wl;

        /* compute yy = gamma(x+1) */
        if (x > 0.2845) {
                if (x > 0.6374) {
                        r1 = x - t0z3;
                        r2 = (double)((float)(r1 - t0z3_l));
                        r2 = (double) ((float) (r1 - t0z3_l));
                        t2 = r1 - r2;
                        yy = GT3(r2, t2 - t0z3_l);
                } else {
                        r1 = x - t0z2;
                        r2 = (double)((float)(r1 - t0z2_l));
                        r2 = (double) ((float) (r1 - t0z2_l));
                        t2 = r1 - r2;
                        yy = GT2(r2, t2 - t0z2_l);
                }
        } else {
                r1 = x - t0z1;
                r2 = (double)((float)(r1 - t0z1_l));
                r2 = (double) ((float) (r1 - t0z1_l));
                t2 = r1 - r2;
                yy = GT1(r2, t2 - t0z1_l);
        }

        /* compute gamma(x+i) = (x+i-1)*...*(x+1)*yy, 0<i<8 */
        switch (i) {
        case 0:                                 /* yy/x */
                r1 = one / x;
                xh = (double)((float)x);        /* x is not tiny */
                rr.h = (double)((float)((yy.h + yy.l) * r1));
                rr.l = r1 * (yy.h - rr.h * xh) - ((r1 * rr.h) * (x - xh) - r1 *
                    yy.l);
                xh = (double) ((float) x);      /* x is not tiny */
                rr.h = (double) ((float) ((yy.h + yy.l) * r1));
                rr.l = r1 * (yy.h - rr.h * xh) -
                        ((r1 * rr.h) * (x - xh) - r1 * yy.l);
                break;
        case 1:                         /* yy */
                rr.h = yy.h;
                rr.l = yy.l;
                break;
        case 2:                         /* (x+1)*yy */
                z = x + one;            /* may not be exact */
```

```
1460                zh = (double)((float)z);
1430                zh = (double) ((float) z);
1461                rr.h = zh * yy.h;
1462                rr.l = z * yy.l + (x - (zh - one)) * yy.h;
1463                break;
1464        case 3:                         /* (x+2)*(x+1)*yy */
1465                z1 = x + one;
1466                z2 = x + 2.0;
1467                z = z1 * z2;
1468                xh = (double)((float)z);
1469                zh = (double)((float)z1);
1438                xh = (double) ((float) z);
1439                zh = (double) ((float) z1);
1470                xl = (x - (zh - one)) * (z2 + zh) - (xh - zh * (zh + one));
1471                rr.h = xh * yy.h;
1472                rr.l = z * yy.l + xl * yy.h;
1473                break;

1475        case 4:                         /* (x+1)*(x+3)*(x+2)*yy */
1476                z1 = x + 2.0;
1477                z2 = (x + one) * (x + 3.0);
1478                zh = z1;
1479                __LO(zh) = 0;
1480                __HI(zh) &= 0xffffff8; /* zh 18 bits mantissa */
1481                zl = x - (zh - 2.0);
1482                z = z1 * z2;
1483                xh = (double)((float)z);
1453                xh = (double) ((float) z);
1484                xl = zl * (z2 + zh * (z1 + zh)) - (xh - zh * (zh * zh - one));
1485                rr.h = xh * yy.h;
1486                rr.l = z * yy.l + xl * yy.h;
1487                break;
1488        case 5:                         /* ((x+1)*(x+4)*(x+2)*(x+3))*yy */
1489                z1 = x + 2.0;
1490                z2 = x + 3.0;
1491                z = z1 * z2;
1492                zh = (double)((float)z1);
1493                yh = (double)((float)z);
1462                zh = (double) ((float) z1);
1463                yh = (double) ((float) z);
1494                yl = (x - (zh - 2.0)) * (z2 + zh) - (yh - zh * (zh + one));
1495                z2 = z - 2.0;
1496                z *= z2;
1497                xh = (double)((float)z);
1467                xh = (double) ((float) z);
1498                xl = yl * (z2 + yh) - (xh - yh * (yh - 2.0));
1499                rr.h = xh * yy.h;
1500                rr.l = z * yy.l + xl * yy.h;
1501                break;
1502        case 6:                         /* ((x+1)*(x+2)*(x+3)*(x+4)*(x+5))*yy */
1503                z1 = x + 2.0;
1504                z2 = x + 3.0;
1505                z = z1 * z2;
1506                zh = (double)((float)z1);
1507                yh = (double)((float)z);
1476                zh = (double) ((float) z1);
1477                yh = (double) ((float) z);
1508                z1 = x - (zh - 2.0);
1509                yl = z1 * (z2 + zh) - (yh - zh * (zh + one));
1510                z2 = z - 2.0;
1511                x5 = x + 5.0;
1512                z *= z2;
1513                xh = (double)((float)z);
1483                xh = (double) ((float) z);
1514                zh += 3.0;
1515                xl = yl * (z2 + yh) - (xh - yh * (yh - 2.0));
```



```
1517                /*
1518                 * xh+xl=(x+1)*...*(x+4)
1519                 * wh+wl=(x+5)*yy
1520                 */
1521                wh = (double)((float)(x5 * (yy.h + yy.l)));
1486                                                /* xh+xl=(x+1)*...*(x+4) */
1487                /* wh+wl=(x+5)*yy */
1488                wh = (double) ((float) (x5 * (yy.h + yy.l)));
1522                wl = (z1 * yy.h + x5 * yy.l) - (wh - zh * yy.h);
1523                rr.h = wh * xh;
1524                rr.l = z * wl + xl * wh;
1525                break;
1526        case 7:                 /* ((x+1)*(x+2)*(x+3)*(x+4)*(x+5)*(x+6))*yy */
1527                z1 = x + 3.0;
1528                z2 = x + 4.0;
1529                z = z2 * z1;
1530                zh = (double)((float)z1);
1531                yh = (double)((float)z);        /* yh+yl = (x+3)(x+4) */
1497                zh = (double) ((float) z1);
1498                yh = (double) ((float) z);      /* yh+yl = (x+3)(x+4) */
1532                yl = (x - (zh - 3.0)) * (z2 + zh) - (yh - (zh * (zh + one)));
1533                z1 = x + 6.0;
1534                z2 = z - 2.0;                   /* z2 = (x+2)*(x+5) */
1535                z *= z2;
1536                xh = (double)((float)z);
1503                xh = (double) ((float) z);
1537                xl = yl * (z2 + yh) - (xh - yh * (yh - 2.0));

1539                /*
1540                 * xh+xl=(x+2)*...*(x+5)
1541                 * wh+wl=(x+1)(x+6)*yy
1542                 */
1505                                                /* xh+xl=(x+2)*...*(x+5) */
1506                /* wh+wl=(x+1)(x+6)*yy */
1543                z2 -= 4.0;              /* z2 = (x+1)(x+6) */
1544                wh = (double)((float)(z2 * (yy.h + yy.l)));
1508                wh = (double) ((float) (z2 * (yy.h + yy.l)));
1545                wl = (z2 * yy.l + yl * yy.h) - (wh - (yh - 6.0) * yy.h);
1546                rr.h = wh * xh;
1547                rr.l = z * wl + xl * wh;
1548        }

1550 #endif /* ! codereview */
1551        return (rr);
1552 }

1554 double
1555 tgamma(double x)
1556 {
1513 tgamma(double x) {
1557        struct Double ss, ww;
1558        double t, t1, t2, t3, t4, t5, w, y, z, z1, z2, z3, z5;
1559        int i, j, k, m, ix, hx, xk;
1560        unsigned lx;

1562        hx = __HI(x);
1563        lx = __LO(x);
1564        ix = hx & 0x7fffffff;
1565        y = x;

1567        if (ix < 0x3ca00000)
1568                return (one / x);       /* |x| < 2**-53 */

1570 #endif /* ! codereview */
1571        if (ix >= 0x7ff00000)
```

```
                /* +Inf -> +Inf, -Inf or NaN -> NaN */
                return (x * ((hx < 0) ? 0.0 : x));

                return (x * ((hx < 0)? 0.0 : x));
        if (hx > 0x406573fa ||          /* x > 171.62... overflow to +inf */
            (hx == 0x406573fa && lx > 0xE561F647)) {
                z = x / tiny;
                return (z * z);
        }

#endif /* ! codereview */
        if (hx >= 0x40200000) {         /* x >= 8 */
                ww = large_gam(x, &m);
                w = ww.h + ww.l;
                __HI(w) += m << 20;
                return (w);
        }

#endif /* ! codereview */
        if (hx > 0) {                   /* 0 < x < 8 */
                i = (int)x;
                ww = gam_n(i, x - (double)i);
                i = (int) x;
                ww = gam_n(i, x - (double) i);
                return (ww.h + ww.l);
        }

        /*
         * negative x
         */

        /* negative x */
        /* INDENT OFF */
        /*
         * compute: xk =
         *      -2 ... x is an even int (-inf is even)
         *      -1 ... x is an odd int
         *      +0 ... x is not an int but chopped to an even int
         *      +1 ... x is not an int but chopped to an odd int
         */
        /* INDENT ON */
        xk = 0;

#endif /* ! codereview */
        if (ix >= 0x43300000) {
                if (ix >= 0x43400000)
                        xk = -2;
                else
                        xk = -2 + (lx & 1);
        } else if (ix >= 0x3ff00000) {
                k = (ix >> 20) - 0x3ff;

#endif /* ! codereview */
                if (k > 20) {
                        j = lx >> (52 - k);

#endif /* ! codereview */
                        if ((j << (52 - k)) == lx)
                                xk = -2 + (j & 1);
                        else
                                xk = j & 1;
                } else {
                        j = ix >> (20 - k);

#endif /* ! codereview */
                        if ((j << (20 - k)) == ix && lx == 0)
```



```
                                xk = -2 + (j & 1);
                        else
                                xk = j & 1;
                }
        }

#endif /* ! codereview */
        if (xk < 0)
                /* ideally gamma(-n)= (-1)**(n+1) * inf, but c99 expect NaN */
                return ((x - x) / (x - x));     /* 0/0 = NaN */


        /* negative underflow thresold */
        if (ix > 0x4066e000 || (ix == 0x4066e000 && lx > 11)) {
                /* x < -183.0 - 11ulp */
                z = tiny / x;

#endif /* ! codereview */
                if (xk == 1)
                        z = -z;

#endif /* ! codereview */
                return (z * tiny);
        }

        /* now compute gamma(x) by  -1/((sin(pi*y)/pi)*gamma(1+y)), y = -x */

        /*
         * First compute ss = -sin(pi*y)/pi , so that
         * gamma(x) = 1/(ss*gamma(1+y))
         */
        y = -x;
        j = (int)y;
        z = y - (double)j;

        if (z > 0.31830988618379067153776 75) {
        j = (int) y;
        z = y - (double) j;
        if (z > 0.31830988618379067153776 75)
                if (z > 0.68169011381620932846223 25)
                        ss = kpsin(one - z);
                else
                        ss = kpcos(0.5 - z);
        } else {
        else
                ss = kpsin(z);
        }

#endif /* ! codereview */
        if (xk == 0) {
                ss.h = -ss.h;
                ss.l = -ss.l;
        }

        /* Then compute ww = gamma(1+y), note that result scale to 2**m */
        m = 0;

#endif /* ! codereview */
        if (j < 7) {
                ww = gam_n(j + 1, z);
        } else {
                w = y + one;

#endif /* ! codereview */
                if ((lx & 1) == 0) {    /* y+1 exact (note that y<184) */
                        ww = large_gam(w, &m);
```

```
                } else {
                        t = w - one;

#endif /* ! codereview */
                        if (t == y) {   /* y+one exact */
                                ww = large_gam(w, &m);
                        } else {        /* use y*gamma(y) */
                                if (j == 7)
                                        ww = gam_n(j, z);
                                else
                                        ww = large_gam(y, &m);

#endif /* ! codereview */
                                t4 = ww.h + ww.l;
                                t1 = (double)((float)y);
                                t2 = (double)((float)t4);
                                t1 = (double) ((float) y);
                                t2 = (double) ((float) t4);
                                /* t4 will not be too large */
                                ww.l = y * (ww.l - (t2 - ww.h)) + (y - t1) * t2;
                                ww.h = t1 * t2;
                        }
                }
        }

        /* compute 1/(ss*ww) */
        t3 = ss.h + ss.l;
        t4 = ww.h + ww.l;
        t1 = (double)((float)t3);
        t2 = (double)((float)t4);
        t1 = (double) ((float) t3);
        t2 = (double) ((float) t4);
        z1 = ss.l - (t1 - ss.h);        /* (t1,z1) = ss */
        z2 = ww.l - (t2 - ww.h);        /* (t2,z2) = ww */
        t3 = t3 * t4;                   /* t3 = ss*ww */
        z3 = one / t3;                  /* z3 = 1/(ss*ww) */
        t5 = t1 * t2;
        z5 = z1 * t4 + t1 * z2;         /* (t5,z5) = ss*ww */
        t1 = (double)((float)t3);       /* (t1,z1) = ss*ww */
        t1 = (double) ((float) t3);     /* (t1,z1) = ss*ww */
        z1 = z5 - (t1 - t5);
        t2 = (double)((float)z3);       /* leading 1/(ss*ww) */
        t2 = (double) ((float) z3);     /* leading 1/(ss*ww) */
        z2 = z3 * (t2 * z1 - (one - t2 * t1));
        z = t2 - z2;

        /* check whether z*2**-m underflow */
        if (m != 0) {
                hx = __HI(z);
                i = hx & 0x80000000;
                ix = hx ^ i;
                j = ix >> 20;

#endif /* ! codereview */
                if (j > m) {
                        ix -= m << 20;
                        __HI(z) = ix ^ i;
                } else if ((m - j) > 52) {
                        /* underflow */
                        if (xk == 0)
                                z = -tiny * tiny;
                        else
                                z = tiny * tiny;
                } else {
                        /* subnormal */
                        m -= 60;
```



```
                        t = one;
                        __HI(t) -= 60 << 20;
                        ix -= m << 20;
                        __HI(z) = ix ^ i;
                        z *= t;
                }
        }

#endif /* ! codereview */
        return (z);
}
```

```
**********************************************************
   15207 Thu Jun  6 20:40:38 2019
new/usr/src/lib/libm/common/m9x/tgammaf.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __tgammaf = tgammaf

/*
 * True gamma function
 *
 * float tgammaf(float x)
 *
 * Algorithm: see tgamma.c
 *
 * Maximum error observed: 0.87ulp (both positive and negative arguments)
 */

#include "libm.h"
#include <math.h>
#if defined(__SUNPRO_C)
#include <sunmath.h>
#endif
#include <sys/isa_defs.h>

#if defined(_BIG_ENDIAN)
#define HIWORD          0
#define LOWORD          1
#else
#define HIWORD          1
#define LOWORD          0
#endif
#define __HI(x)         ((int *)&x)[HIWORD]
#define __LO(x)         ((unsigned *)&x)[LOWORD]
#define __HI(x) ((int *) &x)[HIWORD]
#define __LO(x) ((unsigned *) &x)[LOWORD]
```



```
/* Coefficients for primary intervals GTi() */
static const double cr[] = {
        /* p1 */
        +7.09087253435088360271451613398019280077561279443e-0001,
        -5.17229560788652108545141978238701790105241761089e-0001,
        +5.23403394528150789405825222323770647162337764327e-0001,
        -4.54586308717075010784041566069480411732634814899e-0001,
        +4.20596490915239085459964590559256913498190955233e-0001,
        -3.57307589712377520978332185838241458642142185789e-0001,

        /* p2 */
        +4.28486983980295198166056119223984284434264344578e-0001,
        -1.30704539487709138528680121627899735386650103914e-0001,
        +1.60856285038051955072861219352655851542955430871e-0001,
        -9.22285161346010583774458802067371182158937943507e-0002,
        +7.19240511767225260740890292605070595560626179357e-0002,
        -4.88158265593355093703112238534484636193260459574e-0002,

        /* p3 */
        +3.82409531118807759081121479786092134814808872880e-0001,
        +2.65309888180188647956400403013495759365167853426e-0002,
        +8.06815109775079171923561169415370309376296739835e-0002,
        -1.54821591666137613928840890835174351674007764799e-0002,
        +1.76308239242717268530498313416899188157165183405e-0002,

        /* GZi and TZi */
        +0.938204627909682449409753561580326957698 8,    /* GZ1 */
        +0.885603194410888700278815900582588733208 0,    /* GZ2 */
        +0.936781411463652321618846897080837849742 6,    /* GZ3 */
        -0.351721435785293579101562 5,                   /* TZ1 */
        +0.280530631542205810546875,                     /* TZ3 */
};

#define P10             cr[0]
#define P11             cr[1]
#define P12             cr[2]
#define P13             cr[3]
#define P14             cr[4]
#define P15             cr[5]
#define P20             cr[6]
#define P21             cr[7]
#define P22             cr[8]
#define P23             cr[9]
#define P24             cr[10]
#define P25             cr[11]
#define P30             cr[12]
#define P31             cr[13]
#define P32             cr[14]
#define P33             cr[15]
#define P34             cr[16]
#define GZ1             cr[17]
#define GZ2             cr[18]
#define GZ3             cr[19]
#define TZ1             cr[20]
#define TZ3             cr[21]

/* compute gamma(y) for y in GT1 = [1.0000, 1.2845] */
static double
GT1(double y)
{
GT1(double y) {
        double z, r;

        z = y * y;
        r = TZ1 * y + z * ((P10 + y * P11 + z * P12) + (z * y) * (P13 + y *
```

```
            P14 + z * P15));
        return (GZ1 + r);
}

/* compute gamma(y) for y in GT2 = [1.2844, 1.6374] */
static double
GT2(double y)
{
GT2(double y) {
        double z;

        z = y * y;
        return (GZ2 + z * ((P20 + y * P21 + z * P22) + (z * y) * (P23 + y *
            P24 + z * P25)));
}

/* compute gamma(y) for y in GT3 = [1.6373, 2.0000] */
static double
GT3(double y)
{
        double z, r;
GT3(double y) {
double z, r;

        z = y * y;
        r = TZ3 * y + z * ((P30 + y * P31 + z * P32) + (z * y) * (P33 + y *
            P34));
        return (GZ3 + r);
}

/* INDENT OFF */
static const double c[] = {
        +1.0,
        +2.0,
        +0.5,
        +1.0e-300,
        +6.6667172318485180546936236975392 30e-0001,     /* A1=T3[0] */
        +8.3333333095969406524573688874904281190999457317 8e-0002, /* GP[0] */
        -2.7776554560166717976770660089036153522550776216 8e-0003, /* GP[1] */
        +7.7783085347977528178108527832462103352303748988 3e-0004, /* GP[2] */
        +4.1893853320467274174415078836869577992332032836 9e-0001, /* hln2pi   */
        +2.1660849392498290194 6e-02, /* ln2_32 */
        +4.6166241308446828384 1e+01, /* invln2_32 */
        +5.0000410338898896884115642141566998541407345372 0e-0001, /* Et1 */
        +1.6666765675280076178277827782811020810868754590 8e-0001, /* Et2 */
+1.0,
+2.0,
+0.5,
+1.0e-300,
+6.6667172318485180546936236975392 30e-0001,                    /* A1=T3[0] */
+8.3333333095969406524573688874904281190999457317 8e-0002,       /* GP[0] */
-2.7776554560166717976770660089036153522550776216 8e-0003,       /* GP[1] */
+7.7783085347977528178108527832462103352303748988 3e-0004,       /* GP[2] */
+4.1893853320467274174415078836869577992332032836 9e-0001,       /* hln2pi   */
+2.1660849392498290194 6e-02,                                    /* ln2_32 */
+4.6166241308446828384 1e+01,                                    /* invln2_32 */
+5.0000410338898896884115642141566998541407345372 0e-0001,       /* Et1 */
+1.6666765675280076178277827782811020810868754590 8e-0001,       /* Et2 */
};

#define one             c[0]
#define two             c[1]
#define half            c[2]
#define tiny            c[3]
#define A1              c[4]
#define GP0             c[5]
```



```
#define GP1             c[6]
#define GP2             c[7]
#define hln2pi          c[8]
#define ln2_32          c[9]
#define invln2_32       c[10]
#define Et1             c[11]
#define Et2             c[12]

/* S[j] = 2**(j/32.) for the final computation of exp(w) */
static const double S[] = {
        +1.0000000000000000000e+00,     /* 3FF0000000000000 */
        +1.0218971486541166271 4e+00,   /* 3FF059B0D3158574 */
        +1.0442737824274137548 0e+00,   /* 3FF0B5586CF9890F */
        +1.0671404006768236971 7e+00,   /* 3FF11301D0125B51 */
        +1.0905077326652576896 7e+00,   /* 3FF172B83C7D517B */
        +1.1143867425958924322 1e+00,   /* 3FF1D4873168B9AA */
        +1.1387886347566915645 8e+00,   /* 3FF2387A6E756238 */
        +1.1637248587775774755 2e+00,   /* 3FF29E9DF51FDEE1 */
        +1.1892071150027210269 0e+00,   /* 3FF306FE0A31B715 */
        +1.2152473599804689552 4e+00,   /* 3FF371A7373AA9CB */
        +1.2418578120734840020 1e+00,   /* 3FF3DEA64C123422 */
        +1.2690509571917332198 9e+00,   /* 3FF44E086061892D */
        +1.2968395546510096405 5e+00,   /* 3FF4BFDAD5362A27 */
        +1.3252366431597413232 2e+00,   /* 3FF5342B569D4F82 */
        +1.3542555469368926512 9e+00,   /* 3FF5AB07DD485429 */
        +1.3839098819638320225 8e+00,   /* 3FF6247EB03A5585 */
        +1.4142135623730951454 7e+00,   /* 3FF6A09E667F3BCD */
        +1.4451808069770466502 7e+00,   /* 3FF71F75E8EC5F74 */
        +1.4768261459394993462 3e+00,   /* 3FF7A11473EB0187 */
        +1.5091644275934228414 1e+00,   /* 3FF82589994CCE13 */
        +1.5422108254079407441 1e+00,   /* 3FF8ACE5422AA0DB */
        +1.5759808451078864965 9e+00,   /* 3FF93737B0CDC5E5 */
        +1.6104903319492542834 7e+00,   /* 3FF9C49182A3F090 */
        +1.6457554781539649457 8e+00,   /* 3FFA5503B23E255D */
        +1.6817928305074290040 7e+00,   /* 3FFAE89F995AD3AD */
        +1.7186192981224779341 4e+00,   /* 3FFB7F76F2FB5E47 */
        +1.7562521603732994535 1e+00,   /* 3FFC199BDD85529C */
        +1.7947090750031071682 0e+00,   /* 3FFCB720DCEF9069 */
        +1.8340080864093424306 6e+00,   /* 3FFD5818DCFBA487 */
        +1.8741676341102999625 6e+00,   /* 3FFDFC97337B9B5F */
        +1.9152065613971474000 7e+00,   /* 3FFEA4AFA2A490DA */
        +1.9571441241754001794 1e+00,   /* 3FFF50765B6E4540 */
+1.0000000000000000000e+00,     /* 3FF0000000000000 */
+1.0218971486541166271 4e+00,   /* 3FF059B0D3158574 */
+1.0442737824274137548 0e+00,   /* 3FF0B5586CF9890F */
+1.0671404006768236971 7e+00,   /* 3FF11301D0125B51 */
+1.0905077326652576896 7e+00,   /* 3FF172B83C7D517B */
+1.1143867425958924322 1e+00,   /* 3FF1D4873168B9AA */
+1.1387886347566915645 8e+00,   /* 3FF2387A6E756238 */
+1.1637248587775774755 2e+00,   /* 3FF29E9DF51FDEE1 */
+1.1892071150027210269 0e+00,   /* 3FF306FE0A31B715 */
+1.2152473599804689552 4e+00,   /* 3FF371A7373AA9CB */
+1.2418578120734840020 1e+00,   /* 3FF3DEA64C123422 */
+1.2690509571917332198 9e+00,   /* 3FF44E086061892D */
+1.2968395546510096405 5e+00,   /* 3FF4BFDAD5362A27 */
+1.3252366431597413232 2e+00,   /* 3FF5342B569D4F82 */
+1.3542555469368926512 9e+00,   /* 3FF5AB07DD485429 */
+1.3839098819638320225 8e+00,   /* 3FF6247EB03A5585 */
+1.4142135623730951454 7e+00,   /* 3FF6A09E667F3BCD */
+1.4451808069770466502 7e+00,   /* 3FF71F75E8EC5F74 */
+1.4768261459394993462 3e+00,   /* 3FF7A11473EB0187 */
+1.5091644275934228414 1e+00,   /* 3FF82589994CCE13 */
+1.5422108254079407441 1e+00,   /* 3FF8ACE5422AA0DB */
+1.5759808451078864965 9e+00,   /* 3FF93737B0CDC5E5 */
+1.6104903319492542834 7e+00,   /* 3FF9C49182A3F090 */
+1.6457554781539649457 8e+00,   /* 3FFA5503B23E255D */
```

```
+1.68179283050742900407e+00,    /* 3FFAE89F995AD3AD */
+1.71861929812247793414e+00,    /* 3FFB7F76F2FB5E47 */
+1.75625216037329945351e+00,    /* 3FFC199BDD85529C */
+1.79470907500310716820e+00,    /* 3FFCB720DCEF9069 */
+1.83400808640934243066e+00,    /* 3FFD5818DCFBA487 */
+1.87416763411029996256e+00,    /* 3FFDFC97337B9B5F */
+1.91520656139714740007e+00,    /* 3FFEA4AFA2A490DA */
+1.95714412417540017941e+00,    /* 3FFF50765B6E4540 */
};
/* INDENT ON */
```

```
/* INDENT OFF */
/*
 * return tgammaf(x) in double for 8<x<=35.040096283... using Stirling's formula
 *      log(G(x)) ~= (x-.5)*(log(x)-1) + .5(log(2*pi)-1) + (1/x)*P(1/(x*x))
 */
```

```
#endif /* ! codereview */
/*
 * compute ss = log(x)-1
 *
 *  log(x) - 1 = T1(n) + T2(j) + T3(s), where x = 2**n * y,  1<=y<2,
 *  j=[64*y], z[j]=1+j/64+1/128, s = (y-z[j])/(y+z[j]), and
 *       T1(n-3) = n*log(2)-1,  n=3,4,5
 *       T2(j) = log(z[j]),
 *       T3(s) = 2s + A1*s^3
 *  Note
 *  (1) Remez error for T3(s) is bounded by 2**(-35.8)
 *      (see mpremez/work/Log/tgamma_log_2_outr1)
 */
```

```
static const double T1[] = {            /* T1[j]=(j+3)*log(2)-1 */
        +1.07944154167983592825169636437 5e+00,
        +1.77258872223978123766892848583 3e+00,
        +2.46573590279972654708616060729 1e+00,
+1.07944154167983592825169636437 5e+00,
+1.77258872223978123766892848583 3e+00,
+2.46573590279972654708616060729 1e+00,
};

static const double T2[] = {            /* T2[j]=log(1+j/64+1/128) */
        +7.78214044205494894746290006113 7e-03,
        +2.31670592815343782287991609622 9e-02,
        +3.83188643021365991937553251238 0e-02,
        +5.32445145188122828658701937865 3e-02,
        +6.79506619085077493945652777726 3e-02,
        +8.24436692110745912681600686683 1e-02,
        +9.67296264585511122955710564874 6e-02,
        +1.10814366340290114194806169323 2e-01,
        +1.24703478500957235863406515380 9e-01,
        +1.38402322859119135685325873601 6e-01,
        +1.51916042025841975071803424896 9e-01,
        +1.65249572895307162875611449277 2e-01,
        +1.78407657472818297119400241510 9e-01,
        +1.91394852999629454609298807561 3e-01,
        +2.04215541428690891503820386196 2e-01,
        +2.16873938300614359619089525744 3e-01,
        +2.29374101064845829991480725046 1e-01,
        +2.41719936887145168144307515913 5e-01,
        +2.53915209980963444137323297906 6e-01,
        +2.65963548497137941339125926537 5e-01,
        +2.77868451003456306186350032923 4e-01,
        +2.89633292583042676788930555257 e-01,
        +3.01261330578161781012875538233 8e-01,
        +3.12755710003896888386246559688 3e-01,
```



```
        +3.24119468654211976090670760435 0e-01,
        +3.35355541921137830257179579814 2e-01,
        +3.46466767346208580918462188425 8e-01,
        +3.57455888921803774226009490140 9e-01,
        +3.68325561158707653048230154050 4e-01,
        +3.79078352934969458390853345631 0e-01,
        +3.89716751140025213370463604003 5e-01,
        +4.00243164127012706929325101995 1e-01,
        +4.10659924985268385934306203175 8e-01,
        +4.20969294644129636128867161506 8e-01,
        +4.31173464818371340859172478955 6e-01,
        +4.41274560804875229489496441661 3e-01,
        +4.51274644139458585144692383079 0e-01,
        +4.61175715122170166367999925597 9e-01,
        +4.70979715218791012546897856056 4e-01,
        +4.80688529345751907676618455448 0e-01,
        +4.90303988045193838150346159645 7e-01,
        +4.99827869556449329821331415247 0e-01,
        +5.09261901789807946804074919228 3e-01,
        +5.18607764208045632152976996364 8e-01,
        +5.27867089620842385113892217778 3e-01,
        +5.37041465896883654566729244153 8e-01,
        +5.46132437598135650382397209231 2e-01,
        +5.55141507540501592715480359515 9e-01,
        +5.64070138284802966071384290090 2e-01,
        +5.72919753561785509092756726626 3e-01,
        +5.81691739634622482520610753725 4e-01,
        +5.90387446602176374641916708123 6e-01,
        +5.99008189646083399381600024461 7e-01,
        +6.07555250224541795501085152791 1e-01,
        +6.16029877215514019647565928196 7e-01,
        +6.24433288011893501042538744054 7e-01,
        +6.32766669571037829545786468503 6e-01,
        +6.41031179420931291055601334405 4e-01,
        +6.49227946625109818890839969905 3e-01,
        +6.57358072708360030141890023245 9e-01,
        +6.65422632545090448950092610006 7e-01,
        +6.73422675212166720297960388801 0e-01,
        +6.81359224807903068948071559568 1e-01,
        +6.89233281238808980324914337814 6e-01,
+7.78214044205494894746290006113 7e-03,
+2.31670592815343782287991609622 9e-02,
+3.83188643021365991937553251238 0e-02,
+5.32445145188122828658701937865 3e-02,
+6.79506619085077493945652777726 3e-02,
+8.24436692110745912681600686683 1e-02,
+9.67296264585511122955710564874 6e-02,
+1.10814366340290114194806169323 2e-01,
+1.24703478500957235863406515380 9e-01,
+1.38402322859119135685325873601 6e-01,
+1.51916042025841975071803424896 9e-01,
+1.65249572895307162875611449277 2e-01,
+1.78407657472818297119400241510 9e-01,
+1.91394852999629454609298807561 3e-01,
+2.04215541428690891503820386196 2e-01,
+2.16873938300614359619089525744 3e-01,
+2.29374101064845829991480725046 1e-01,
+2.41719936887145168144307515913 5e-01,
+2.53915209980963444137323297906 6e-01,
+2.65963548497137941339125926537 5e-01,
+2.77868451003456306186350032923 4e-01,
+2.89633292583042676788930555257 e-01,
+3.01261330578161781012875538233 8e-01,
+3.12755710003896888386246559688 3e-01,
+3.24119468654211976090670760435 0e-01,
+3.35355541921137830257179579814 2e-01,
```

```
223 +3.464667673462085809184621884258e-01,
224 +3.574558889218037742260094901409e-01,
225 +3.683255611587076530482301540504e-01,
226 +3.790783529349694583908533456310e-01,
227 +3.897167511400252133704636040035e-01,
228 +4.002431641270127069293251019951e-01,
229 +4.106599249852683859343062031758e-01,
230 +4.209692946441296361288671615068e-01,
231 +4.311734648183713408591724789556e-01,
232 +4.412745608048752294894964416613e-01,
233 +4.512746441394585851446923830790e-01,
234 +4.611757151221701663679999255979e-01,
235 +4.709797152187910125468978560564e-01,
236 +4.806885293457519076766184554480e-01,
237 +4.903039880451938381503461596457e-01,
238 +4.998278695564493298213314152470e-01,
239 +5.092619017898079468040749192283e-01,
240 +5.186077642080456321529769963648e-01,
241 +5.278670896208423851138922177783e-01,
242 +5.370414658968836545667292441538e-01,
243 +5.461324375981356503823972092312e-01,
244 +5.551415075405015927154803595159e-01,
245 +5.640701382848029660713842900902e-01,
246 +5.729197535617855090927567266263e-01,
247 +5.816917396346224825206107537254e-01,
248 +5.903874466021763746419167081236e-01,
249 +5.990081896460833993816000244617e-01,
250 +6.075555250224541795501085152791 1e-01,
251 +6.160298772155140196475659281967e-01,
252 +6.244332880118935010425387440547e-01,
253 +6.327666695710378295457864685036e-01,
254 +6.410311794209312910556013344054e-01,
255 +6.492279466251098188908399699053e-01,
256 +6.573580727083600301418900232459e-01,
257 +6.654226325450904489500926100067e-01,
258 +6.734226752121667202979603888010e-01,
259 +6.813592248079030689480715595681e-01,
260 +6.892332812388089803249143378146e-01,
305 };
262 /* INDENT ON */

307 static double
308 large_gam(double x)
309 {
265 large_gam(double x) {
310         double ss, zz, z, t1, t2, w, y, u;
311         unsigned lx;
312         int k, ix, j, m;

314         ix = __HI(x);
315         lx = __LO(x);
316         m = (ix >> 20) - 0x3ff;                 /* exponent of x, range:3-5 */
317         ix = (ix & 0x000fffff) | 0x3ff00000;    /* y = scale x to [1,2] */
318         __HI(y) = ix;
319         __LO(y) = lx;
320         __HI(z) = (ix & 0xffffc000) | 0x2000;   /* z[j]=1+j/64+1/128 */
321         __LO(z) = 0;
322         j = (ix >> 14) & 0x3f;
323         t1 = y + z;
324         t2 = y - z;
325         u = t2 / t1;
326         ss = T1[m - 3] + T2[j] + u * (two + A1 * (u * u));
327         /* ss = log(x)-1 */

329 #endif /* ! codereview */
330         /*
```



```
331          * compute ww = (x-.5)*(log(x)-1) + .5*(log(2pi)-1) + 1/x*(P(1/x^2)))
332          * where ss = log(x) - 1
333          */
334         z = one / x;
335         zz = z * z;
336         w = ((x - half) * ss + hln2pi) + z * (GP0 + zz * GP1 + (zz * zz) * GP2);
337         k = (int)(w * invln2_32 + half);
284         k = (int) (w * invln2_32 + half);

339         /* compute the exponential of w */
340         j = k & 0x1f;
341         m = k >> 5;
342         z = w - (double)k * ln2_32;
289         z = w - (double) k *ln2_32;
343         zz = S[j] * (one + z + (z * z) * (Et1 + z * Et2));
344         __HI(zz) += m << 20;
345         return (zz);
346 }


294 /* INDENT OFF */
349 /*
350  * kpsin(x)= sin(pi*x)/pi
351  *                3          5         7         9
352  *      = x+ks[0]*x +ks[1]*x +ks[2]*x +ks[3]*x
353  */
354 static const double ks[] = {
355         -1.644934049856458113544766650520053428394477905 44e+0000,
356         +8.117407944583510640927972490694382693673892722 70e-0001,
357         -1.907031446035512169330758091628895368788540552 02e-0001,
358         +2.557423339942645632811553122714811086355753312 01e-0002,
301 -1.644934049856458113544766650520053428394477905 44e+0000,
302 +8.117407944583510640927972490694382693673892722 70e-0001,
303 -1.907031446035512169330758091628895368788540552 02e-0001,
304 +2.557423339942645632811553122714811086355753312 01e-0002,
359 };
306 /* INDENT ON */

361 static double
362 kpsin(double x)
363 {
309 kpsin(double x) {
364         double z;

366         z = x * x;
367         return (x + (x * z) * ((ks[0] + z * ks[1]) + (z * z) * (ks[2] + z *
368             ks[3])));
369 }


317 /* INDENT OFF */
372 /*
373  * kpcos(x)= cos(pi*x)/pi
374  *                2         4         6
375  *      = kc[0]+kc[1]*x +kc[2]*x +kc[3]*x
376  */
377 static const double kc[] = {
378         +3.183098861837906715377675267450287240689192914 80e-0001,
379         -1.570795814477625681994678750658545386265949377 91e+0000,
380         +1.291835280925586928440730040295686740278073938 62e+0000,
381         -4.202329497713076859810159144251954716027390755 37e-0001,
324 +3.183098861837906715377675267450287240689192914 80e-0001,
325 -1.570795814477625681994678750658545386265949377 91e+0000,
326 +1.291835280925586928440730040295686740278073938 62e+0000,
327 -4.202329497713076859810159144251954716027390755 37e-0001,
382 };
```

```
329 /* INDENT ON */

384 static double
385 kpcos(double x)
386 {
332 kpcos(double x) {
387         double z;

389         z = x * x;
390         return (kc[0] + z * (kc[1] + z * kc[2] + (z * z) * kc[3]));
391 }

393 static const double t0z1 = 0.134861805732790769689793935774652917006,
394         t0z2 = 0.461632144968362341262659542325721328468,
395         t0z3 = 0.819773101100500601787868704921606996312;

397 /*
398  * 1.134861805732790769689793935774652917006
399  */
339 /* INDENT OFF */
340 static const double
341 t0z1 = 0.134861805732790769689793935774652917006,
342 t0z2 = 0.461632144968362341262659542325721328468,
343 t0z3 = 0.819773101100500601787868704921606996312;
344         /* 1.134861805732790769689793935774652917006 */
345 /* INDENT ON */

401 /*
402  * gamma(x+i) for 0 <= x < 1
403  */
404 static double
405 gam_n(int i, double x)
406 {
351 gam_n(int i, double x) {
407         double rr = 0.0L, yy;
408         double z1, z2;

410         /* compute yy = gamma(x+1) */
411         if (x > 0.2845) {
412                 if (x > 0.6374)
413                         yy = GT3(x - t0z3);
414                 else
415                         yy = GT2(x - t0z2);
416         } else {
361         } else
417                 yy = GT1(x - t0z1);
418         }
419 #endif /* ! codereview */

421         /* compute gamma(x+i) = (x+i-1)*...*(x+1)*yy, 0<i<8 */
422         switch (i) {
423         case 0:                         /* yy/x */
424                 rr = yy / x;
425                 break;
426         case 1:                         /* yy */
427                 rr = yy;
428                 break;
429         case 2:                         /* (x+1)*yy */
430                 rr = (x + one) * yy;
431                 break;
432         case 3:                         /* (x+2)*(x+1)*yy */
433                 rr = (x + one) * (x + two) * yy;
434                 break;

436         case 4:                         /* (x+1)*(x+3)*(x+2)*yy */
437                 rr = (x + one) * (x + two) * ((x + 3.0) * yy);
```



```
438                 break;
439         case 5:                         /* ((x+1)*(x+4)*(x+2)*(x+3))*yy */
440                 z1 = (x + two) * (x + 3.0) * yy;
441                 z2 = (x + one) * (x + 4.0);
442                 rr = z1 * z2;
443                 break;
444         case 6:                         /* ((x+1)*(x+2)*(x+3)*(x+4)*(x+5))*yy */
445                 z1 = (x + two) * (x + 3.0);
446                 z2 = (x + 5.0) * yy;
447                 rr = z1 * (z1 - two) * z2;
448                 break;
449         case 7:                 /* ((x+1)*(x+2)*(x+3)*(x+4)*(x+5)*(x+6))*yy */
450                 z1 = (x + two) * (x + 3.0);
451                 z2 = (x + 5.0) * (x + 6.0) * yy;
452                 rr = z1 * (z1 - two) * z2;
453                 break;
454         }

456 #endif /* ! codereview */
457         return (rr);
458 }

460 float
461 tgammaf(float xf)
462 {
363 tgammaf(float xf) {
463         float zf;
464         double ss, ww;
465         double x, y, z;
466         int i, j, k, ix, hx, xk;

468         hx = *(int *)&xf;
369         hx = *(int *) &xf;
469         ix = hx & 0x7fffffff;

471         x = (double)xf;

372         x = (double) xf;
473         if (ix < 0x33800000)
474                 return (1.0F / xf);                     /* |x| < 2**-24 */

476         if (ix >= 0x7f800000)
477                 return (xf * ((hx < 0) ? 0.0F : xf));   /* +-Inf or NaN */
377                 return (xf * ((hx < 0)? 0.0F : xf)); /* +-Inf or NaN */

479         if (hx > 0x420C290F)    /* x > 35.040096283... overflow */
480                 return ((float)(x / tiny));
380                 return (float)(x / tiny);

482         if (hx >= 0x41000000)                           /* x >= 8 */
483                 return ((float)large_gam(x));
383                 return ((float) large_gam(x));

485         if (hx > 0) {                                   /* 0 < x < 8 */
486                 i = (int)xf;
487                 return ((float)gam_n(i, x - (double)i));
386                 i = (int) xf;
387                 return ((float) gam_n(i, x - (double) i));
488         }

490         /*
491          * negative x
492          */

390         /* negative x */
391         /* INDENT OFF */
```

```
494         /*
495          * compute xk =
496          *      -2 ... x is an even int (-inf is considered even)
497          *      -1 ... x is an odd int
498          *      +0 ... x is not an int but chopped to an even int
499          *      +1 ... x is not an int but chopped to an odd int
500          */
399         /* INDENT ON */
501         xk = 0;

503 #endif /* ! codereview */
504         if (ix >= 0x4b000000) {
505                 if (ix > 0x4b000000)
506                         xk = -2;
507                 else
508                         xk = -2 + (ix & 1);
509         } else if (ix >= 0x3f800000) {
510                 k = (ix >> 23) - 0x7f;
511                 j = ix >> (23 - k);

513 #endif /* ! codereview */
514                 if ((j << (23 - k)) == ix)
515                         xk = -2 + (j & 1);
516                 else
517                         xk = j & 1;
518         }

520 #endif /* ! codereview */
521         if (xk < 0) {
522                 /* 0/0 invalid NaN, ideally gamma(-n)= (-1)**(n+1) * inf */
523                 zf = xf - xf;
524                 return (zf / zf);
525         }

527         /* negative underflow thresold */
528         if (ix > 0x4224000B) {          /* x < -(41+11ulp) */
529                 if (xk == 0)
530                         z = -tiny;
531                 else
532                         z = tiny;

534 #endif /* ! codereview */
535                 return ((float)z);
536         }

538         /*
539          * now compute gamma(x) by  -1/((sin(pi*y)/pi)*gamma(1+y)), y = -x
540          */

401         /* INDENT OFF */
402         /* now compute gamma(x) by  -1/((sin(pi*y)/pi)*gamma(1+y)), y = -x */
542         /*
543          * First compute ss = -sin(pi*y)/pi , so that
544          * gamma(x) = 1/(ss*gamma(1+y))
545          */
407         /* INDENT ON */
546         y = -x;
547         j = (int)y;
548         z = y - (double)j;

550         if (z > 0.31830988618379067153776 75) {
409         j = (int) y;
410         z = y - (double) j;
411         if (z > 0.31830988618379067153776 75)
551                 if (z > 0.68169011381620932846223 25)
552                         ss = kpsin(one - z);
```



```
553                 else
554                         ss = kpcos(0.5 - z);
555         } else {
416         else
556                 ss = kpsin(z);
557         }

559 #endif /* ! codereview */
560         if (xk == 0)
561                 ss = -ss;

563         /* Then compute ww = gamma(1+y)  */
564         if (j < 7)
565                 ww = gam_n(j + 1, z);
566         else
567                 ww = large_gam(y + one);

569         /* return 1/(ss*ww) */
570         return ((float)(one / (ww * ss)));
418         return ((float) (one / (ww * ss)));
571 }
```

_____*unchanged_portion_omitted_*

```
**********************************************************
   39809 Thu Jun  6 20:40:38 2019
new/usr/src/lib/libm/common/m9x/tgammal.c
11210 libm should be cstyle(10NBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __tgammal = tgammal

#include "libm.h"
#include <sys/isa_defs.h>

#if defined(_BIG_ENDIAN)
#define H0_WORD(x)              ((unsigned *)&x)[0]
#define H3_WORD(x)              ((unsigned *)&x)[3]
#define CHOPPED(x)              (long double)((double)(x))
#define H0_WORD(x)      ((unsigned *) &x)[0]
#define H3_WORD(x)      ((unsigned *) &x)[3]
#define CHOPPED(x)      (long double) ((double) (x))
#else
#define H0_WORD(x)              ((((int *)&x)[2] << 16) | (0x0000ffff & \
        (((unsigned *)&x)[1] >> 15)))
#define H3_WORD(x)              ((unsigned *)&x)[0]
#define CHOPPED(x)              (long double)((float)(x))
#define H0_WORD(x)      ((((int *) &x)[2] << 16) | \
                        (0x0000ffff & (((unsigned *) &x)[1] >> 15)))
#define H3_WORD(x)      ((unsigned *) &x)[0]
#define CHOPPED(x)      (long double) ((float) (x))
#endif

struct LDouble {
        long double h, l;
};

/*
 * Primary interval GTi()
 */
```



```
/* INDENT OFF */
/* Primary interval GTi() */
static const long double P1[] = {
        +0.7090868361997779190371857415076101246115137205 57L,
        +4.4575478120648903582791596936735483566739160695 1e-0001L,
        +3.2104929873583238231166227388263221006291815385 2e-0002L,
        -5.7129679634210661765176524585828919736968886435 50e-0003L,
        +6.0466689289199897708161917496985583160696535277 3e-0003L,
        +8.9910618699688871193962781217476525882265864516 8e-0004L,
        -6.9649684614440774143120700852701844181017556894 9e-0005L,
        +1.5259704611898402081422540930013144507021388242 9e-0005L,
        +5.6852107616849567384471146540743218919068154154 7e-0007L,
        +3.3074967351963489522058206252028656561041895297 9e-0008L,
};

#endif /* ! codereview */
static const long double Q1[] = {
        +1.0 + 0000L,
        +1.0+0000L,
        +1.3580651172167107040857085353725707957949065066 8e+0000L,
        +2.9756781015342955340532714009606308699407295296 1e-0001L,
        -1.5295683598258857150295437282168185168111809787 0e-0001L,
        -2.8824851956142010976878161528908205359795452121 8e-0002L,
        +1.0347531171993740521978994845631393630237839595 5e-0002L,
        +4.1231020324389122236896536012439129737482274231 3e-0004L,
        -3.1265370815229086724893192512038072951833250738 8e-0004L,
        +2.3667217085040974523735810566775776052701433245 8e-0005L,
};

#endif /* ! codereview */
static const long double P2[] = {
        +0.4284868158555854297302099078106501352552706006680841 14L,
        +2.6276847910380976280569174330542407797523055117 6e-0001L,
        +3.8118753268539229760831083799519394659142589615 0e-0002L,
        +3.0006307589181104382066684612913125594852792538 1e-0003L,
        +2.4731540781227916422839847079749864914251340865 4e-0003L,
        +3.6283819991784837258617348314721488046478293866 4e-0004L,
        +3.4399110597549262398272564404647303009817269242 3e-0006L,
        +4.5690215156960327223701424079425765915904543289 5e-0006L,
        +2.1373475583759569560204510067554001135294895845 3e-0007L,
        +9.7412344054791823078167026696788249223487712535 8e-0009L,
};

#endif /* ! codereview */
static const long double Q2[] = {
        +1.0L,
        +9.1828411863250684266464551683076148970055617970 1e-0001L,
        -6.4143085883783076604520207696592377618915487494 7e-0003L,
        -1.2440088580977107321334574743796414977541092137 6e-0001L,
        +4.6980379814625175753885656752248197962474687596 4e-0003L,
        +7.1830944706949531591428470510986869626266208273 1e-0003L,
        -8.7581262698789469511272260069765342578616639910 5e-0004L,
        -1.2353997237776927799595933918843149862667483516 9e-0004L,
        +3.1001901759015159873236009784967292544858754774 6e-0005L,
        -1.7726022334933261765892187428802677746578236407 0e-0006L,
};

#endif /* ! codereview */
static const long double P3[] = {
        +0.3824094797345675048502747661075355640070439388 902L,
        +3.4219809307661849541585490633590842715983337777 4e-0001L,
        +9.6382818950058556830396140686315323744070275485 8e-0002L,
        +8.7606942104269638485246204418852025215684676866 7e-0003L,
        +1.8647789038916149122487201414930901526189753748 8e-0003L,
        +8.1687135454030989587997474285370131154128694419 1e-0004L,
        +6.8378348367460032251869509086465938165012562521 6e-0005L,
```

```
        -1.101682697192615747085659351727192092721908284 56e-0006L,
        +9.662432285083804201592348532789067170656297210 16e-0007L,
        +2.318588855791772505411638206711216649743347281 42e-0008L,
};

#endif /* ! codereview */
static const long double Q3[] = {
        +1.0L, +8.254798211688136346324374300903762525127930673 39e-0001L,
        +1.0L,
        +8.254798211688136346324374300903762525127930673 39e-0001L,
        -1.622513630739377697396396236692951103460155763 20e-0002L,
        -1.106212869059167327587451306294265596911875798 52e-0001L,
        +3.483096939709856126444464157892300155153652914 59e-0003L,
        +6.735537374874883330324312611312896723470434013 28e-0003L,
        -7.632220083933726301627435878110046130502451280 51e-0004L,
        -1.357926706691906314767847689619537117730732513 36e-0004L,
        +3.196101509542235870062207300656081564602056906 18e-0005L,
        -1.820965538628223466101095220151295856933543483 22e-0006L,
};

static const long double
#if defined(__x86)
        GZ1_h = 0.938204627909682449364570100414084663498215377L,
        GZ1_l = 4.518346116624229420055327632718530617227944106e-20L,
        GZ2_h = 0.885603194410888700264725126309883762587560340L,
        GZ2_l = 1.409077427270497062039119290776508217077297169e-20L,
        GZ3_h = 0.936781411463652321613537060640553022494714241L,
        GZ3_l = 5.309836440284827247897772963887219035221996813e-21L,
GZ1_h   =  0.938204627909682449364570100414084663498215377L,
GZ1_l   =  4.518346116624229420055327632718530617227944106e-20L,
GZ2_h   =  0.885603194410888700264725126309883762587560340L,
GZ2_l   =  1.409077427270497062039119290776508217077297169e-20L,
GZ3_h   =  0.936781411463652321613537060640553022494714241L,
GZ3_l   =  5.309836440284827247897772963887219035221996813e-21L,
#else
        GZ1_h = 0.938204627909682449409753561580326910854647031L,
        GZ1_l = 4.684412162199460089642452580902345976446297037e-35L,
        GZ2_h = 0.885603194410888700278815900582588658192658794L,
        GZ2_l = 7.501529273890253789219935569758713534641074860e-35L,
        GZ3_h = 0.936781411463652321618846897080837818855399840L,
        GZ3_l = 3.088721217404784363585591914529361687403776917e-35L,
GZ1_h   =  0.938204627909682449409753561580326910854647031L,
GZ1_l   =  4.684412162199460089642452580902345976446297037e-35L,
GZ2_h   =  0.885603194410888700278815900582588658192658794L,
GZ2_l   =  7.501529273890253789219935569758713534641074860e-35L,
GZ3_h   =  0.936781411463652321618846897080837818855399840L,
GZ3_l   =  3.088721217404784363585591914529361687403776917e-35L,
#endif
        TZ1 = -0.351721435785293579101562 5L,
        TZ3 = 0.280530631542205810546875L;

TZ1     = -0.351721435785293579101562 5L,
TZ3     =  0.280530631542205810546875L;
/* INDENT ON */

/* INDENT OFF */
/*
 * compute gamma(y=yh+yl) for y in GT1 = [1.0000, 1.2845]
 * ...assume yh got 53 or 24(i386) significant bits
 */
/* INDENT ON */
static struct LDouble
GT1(long double yh, long double yl)
{
GT1(long double yh, long double yl) {
        long double t3, t4, y;
```



```
        int i;
        struct LDouble r;

        y = yh + yl;

#endif /* ! codereview */
        for (t4 = Q1[8], t3 = P1[8] + y * P1[9], i = 7; i >= 0; i--) {
                t4 = t4 * y + Q1[i];
                t3 = t3 * y + P1[i];
        }

#endif /* ! codereview */
        t3 = (y * y) * t3 / t4;
        t3 += (TZ1 * yl + GZ1_l);
        t4 = TZ1 * yh;
        r.h = CHOPPED((t4 + GZ1_h + t3));
        t3 += (t4 - (r.h - GZ1_h));
        r.l = t3;
        return (r);
}


/* INDENT OFF */
/*
 * compute gamma(y=yh+yl) for y in GT2 = [1.2844, 1.6374]
 * ...assume yh got 53 significant bits
 */
/* INDENT ON */
static struct LDouble
GT2(long double yh, long double yl)
{
GT2(long double yh, long double yl) {
        long double t3, t4, y;
        int i;
        struct LDouble r;

        y = yh + yl;

#endif /* ! codereview */
        for (t4 = Q2[9], t3 = P2[9], i = 8; i >= 0; i--) {
                t4 = t4 * y + Q2[i];
                t3 = t3 * y + P2[i];
        }

#endif /* ! codereview */
        t3 = GZ2_l + (y * y) * t3 / t4;
        r.h = CHOPPED((GZ2_h + t3));
        r.l = t3 - (r.h - GZ2_h);
        return (r);
}


/* INDENT OFF */
/*
 * compute gamma(y=yh+yl) for y in GT3 = [1.6373, 2.0000]
 * ...assume yh got 53 significant bits
 */
/* INDENT ON */
static struct LDouble
GT3(long double yh, long double yl)
{
GT3(long double yh, long double yl) {
        long double t3, t4, y;
        int i;
        struct LDouble r;
```

```
224         y = yh + yl;

226 #endif /* ! codereview */
227         for (t4 = Q3[9], t3 = P3[9], i = 8; i >= 0; i--) {
228                 t4 = t4 * y + Q3[i];
229                 t3 = t3 * y + P3[i];
230         }

232 #endif /* ! codereview */
233         t3 = (y * y) * t3 / t4;
234         t3 += (TZ3 * yl + GZ3_l);
235         t4 = TZ3 * yh;
236         r.h = CHOPPED((t4 + GZ3_h + t3));
237         t3 += (t4 - (r.h - GZ3_h));
238         r.l = t3;
239         return (r);
240 }

242 /*
243  * Hex value of GP[0] shoule be 3FB55555 55555555
244  */
134 /* INDENT OFF */
135 /* Hex value of GP[0] shoule be 3FB55555 55555555 */
245 static const long double GP[] = {
246         +0.083333333333333333333333333333333172839171301L,
247         -2.777777777777777777777777749250121199939942410 4e-0003L,
248         +7.936507936507936507936356505416382363500208832 43e-0004L,
249         -5.952380952380952380572997726793245033392419617 04e-0004L,
250         +8.417508417508416961384229879776835249261426003 21e-0004L,
251         -1.917526917526866828250325478236996621788421233 08e-0003L,
252         +6.410256410224034809218915593564734511612793593 22e-0003L,
253         -2.955065357984140191898195874555770037328081850 71e-0002L,
254         +1.796443672299700314860791800609230734765687321 36e-0001L,
255         -1.392430864872746621745628725670572002556492906 46e+0000L,
256         +1.340258740444179621886778164778422652596082697 75e+0001L,
257         -1.568037134801274694144955453999825087007482743 18e+0002L,
258         +2.187398416562015616949276303350993139689244938 91e+0003L,
259         -3.552498486441003384191870380909254109762379212 69e+0004L,
260         +6.434648804378352862167689594394843764491795764 52e+0005L,
261         -1.204591543855770149926003427828213896058939046 24e+0007L,
262         +2.092632496373512985639349423497497184910710932 10e+0008L,
263         -2.962474831831692193437453164338995998346857034 57e+0009L,
264         +2.889849336058960331547276260865067569723272929 81e+0010L,
265         -1.409604341460300077328383824162306103026780639 84e+0011L, /* 19 */
266 };
```

*unchanged_portion_omitted_*

```
277 /* BEGIN CSTYLED */

278 static const long double c[] = {
279         0.0L,
280         1.0L,
281         2.0L,
282         0.5L,
283         1.0e-4930L,                     /* tiny */
284         4.189376831054687500 00e-01L, /* hln2pim1_h */
285         8.500992039917803297364056176398613974736377834128 17152e-07L, /* hln2pim
286         0.418938533204672741780329736405617639861397473637783412817152L, /* hln2
287         2.166084938653511926531791687011718 75e-02L, /* ln2_32hi */
288         5.963171653970586925450830252359379198757976691271 30e-12L, /* ln2_32lo *
289         46.166241308446829035517589792060548397652670532895549892 33L, /* invln2_
170 0.0L,
171 1.0L,
172 2.0L,
173 0.5L,
174 1.0e-4930L,                                                 /* tiny */
175 4.189376831054687500 00e-01L,                               /* hln2pim1_h */
```



```
176 8.500992039917803297364056176398613974736377834128 17152e-07L,   /* hln2pim1_l */
177 0.418938533204672741780329736405617639861397473637783412817152L, /* hln2pim1 */
178 2.166084938653511926531791687011718 75e-02L,                     /* ln2_32hi */
179 5.963171653970586925450830252359379198757976691271 30e-12L,      /* ln2_32lo */
180 46.166241308446829035517589792060548397652670532895549892 33L,   /* invln2_32 */
290 #if defined(__x86)
291         1.755548342904462917002383903763984562829 1e+03L, /* overflow */
182 1.755548342904462917002383903763984562829 1e+03L,                /* overflow */
292 #else
293         1.755548342904462917003889216070203203417 7e+03L, /* overflow */
184 1.755548342904462917003889216070203203417 7e+03L,                /* overflow */
294 #endif
295 };
296 /* END CSTYLED */
297 #endif /* ! codereview */

299 #define zero                    c[0]
300 #define one                     c[1]
301 #define two                     c[2]
302 #define half                    c[3]
303 #define tiny                    c[4]
304 #define hln2pim1_h              c[5]
305 #define hln2pim1_l              c[6]
306 #define hln2pim1                c[7]
307 #define ln2_32hi                c[8]
308 #define ln2_32lo                c[9]
309 #define invln2_32               c[10]
310 #define overflow                c[11]

312 /*
313  * |exp(r) - (1+r+Et0*r^2+...+Et10*r^12)| <= 2^(-128.88) for |r|<=ln2/64
314  */
315 static const long double Et[] = {
316         +5.000000000000000000e-1L,
317         +1.666666666666666666666666666668288351662921524 66e-0001L,
318         +4.166666666666666666666666666666933986465927121 89e-0002L,
319         +8.333333333333333333333317487745126017755911159 51e-0003L,
320         +1.388888888888888888888884535601151139476475399 7e-0003L,
321         +1.984126984126984132371403500929932526841988821 02e-0004L,
322         +2.480158730158730160802220253574426598958143716 94e-0005L,
323         +2.755731922390289211145729864419721409334323177 98e-0006L,
324         +2.755731922394484705555481028955263697398562193 17e-0007L,
325         +2.505216778676839359408539979959376002141672324 77e-0008L,
326         +2.087679288990103673749844485136855665141521473 62e-0009L,
327 };

329 /*
330  * long double precision coefficients for computing log(x)-1 in tgamma.
331  *  See "algorithm" for details
332  *
333  *  log(x) - 1 = T1(n) + T2(j) + T3(s), where x = 2**n * y,  1<=y<2,
334  *  j=[64*y], z[j]=1+j/64+1/128, s = (y-z[j])/(y+z[j]), and
335  *       T1(n) = T1[2n,2n+1] = n*log(2)-1,
336  *       T2(j) = T2[2j,2j+1] = log(z[j]),
337  *       T3(s) = 2s + T3[0]s^3 + T3[1]s^5 + T3[2]s^7 + ... + T3[6]s^15
338  *  Note
339  *  (1) the leading entries are truncated to 24 binary point.
340  *  (2) Remez error for T3(s) is bounded by 2**(-136.54)
341  */
342 static const long double T1[] = {
343         -1.000000000000000000000000000000000000000e+00L,
187 -1.000000000000000000000000000000000000000e+00L,
344         +0.000000000000000000000000000000000000000e+00L,
345         -3.068528175354003906250000000000000000000e-01L,
346         -1.904654299957767878541823431924500011926579e-09L,
189 -3.068528175354003906250000000000000000000e-01L,
```

```
 190 -1.904654299957767878541823431924500011926579e-09L,
 347         +3.862943053245544433593750000000000000000000e-01L,
 348         +5.579533617547508924291635313615100141107647e-08L,
 349         +1.079441487789154052734375000000000000000000e+00L,
 350         +5.389068187551732136437452970422650211661470e-08L,
 351         +1.772588670253753662109375000000000000000000e+00L,
 352         +5.198602757555955348583270627230200282215294e-08L,
 353         +2.465735852718353271484375000000000000000000e+00L,
 354         +5.008137327560178560729088284037750352769117e-08L,
 355         +3.158883035182952880859375000000000000000000e+00L,
 356         +4.817671897564401772874905940845299849351090e-08L,
 357         +3.852030217647552490234375000000000000000000e+00L,
 358         +4.627206467568624985020723597652849919904913e-08L,
 359         +4.545177400112152099609375000000000000000000e+00L,
 360         +4.436741037572848197166541254460399990458737e-08L,
 361         +5.238324582576751708984375000000000000000000e+00L,
 362         +4.246275607577071409312358911267950061012560e-08L,
 363         +5.931471765041351318359375000000000000000000e+00L,
 364         +4.055810177581294621458176568075500131566384e-08L,
 365 };
_____unchanged_portion_omitted_

 574 #endif /* ! codereview */
 575 static const long double S_trail[] = {
 576 #if defined(__x86)
 577         +0.000000000000000000000000e+00L,
 578         +2.632796566718088256938252 4e-20L,
 579         +8.376586352189519112966189 9e-20L,
 580         +3.979870577745404249209575e-20L,
 581         +1.066804659665155864099304 2e-19L,
 582         +1.937600984728536044811711 4e-20L,
 583         +6.708181945611295375127757 6e-21L,
 584         +1.971168050262918646272972 7e-20L,
 585         +2.993258443844952368910456 9e-20L,
 586         +6.888775415303910941106191 4e-20L,
 587         +6.800271874122537894284782 0e-20L,
 588         +6.584691737697540343974234 9e-20L,
 589         +1.217195872751137219487600 1e-20L,
 590         +3.562525322870408711543826 0e-20L,
 591         +3.112955155907756095630917 9e-20L,
 592         +5.751919239616477984621649 2e-20L,
 593         +3.790065117786514159310123 9e-20L,
 594         +1.165926240569874179808011 5e-20L,
 595         +7.136438510284695967172478e-20L,
 596         +5.263100371081220358878894 9e-20L,
 597         +2.632885378873263286846058 0e-20L,
 598         +5.458395008543824278819014 1e-20L,
 599         +9.580325437693826996071865 6e-20L,
 600         +7.683773398387424582351227 9e-21L,
 601         +2.441596591083509382420208 7e-20L,
 602         +2.605296687101658098176972 8e-20L,
 603         +2.687645634463255387530957 9e-21L,
 604         +1.286193015561370020170327 9e-20L,
 605         +8.816663339403748560657229 4e-20L,
 606         +2.978861538958019094083703 7e-20L,
 607         +5.235234161980509867742213 9e-20L,
 608         +5.257846306401046373224236 3e-20L,
 609 #else
 610         +0.000000000000000000000000000000000000e+00L,
 611         +1.805067874203309547455733330545737 86e-35L,
 612         -9.374520292280427421957567419730832 14e-35L,
 613         -1.596968447292758770712909630231499 97e-35L,
 417 -9.374520292280427421957567419730832 14e-35L,
 418 -1.596968447292758770712909630231499 97e-35L,
 614         +9.112493410125022978511686101672486 66e-35L,
 615         -6.504228206978548287230374775259388 71e-35L,
```



```
 616         -8.148468844525851137325691767488155 32e-35L,
 617         -5.066214576721800313372330745142903 35e-35L,
 618         -1.359830974688816973749875638245919 12e-35L,
 420 -6.504228206978548287230374775259388 71e-35L,
 421 -8.148468844525851137325691767488155 32e-35L,
 422 -5.066214576721800313372330745142903 35e-35L,
 423 -1.359830974688816973749875638245919 12e-35L,
 619         +9.497427635563196470307710566433246 60e-35L,
 620         -3.283170523176998601615065965333915 26e-36L,
 621         -5.017235709387190410290186530458428 95e-35L,
 622         -2.391474797689109171622834301602640 14e-35L,
 623         -8.350571357633908815298890737944083 85e-36L,
 425 -3.283170523176998601615065965333915 26e-36L,
 426 -5.017235709387190410290186530458428 95e-35L,
 427 -2.391474797689109171622834301602640 14e-35L,
 428 -8.350571357633908815298890737944083 85e-36L,
 624         +7.036756889073265042421737190671876 44e-35L,
 625         -5.182484853064646457536893018566956 19e-35L,
 430 -5.182484853064646457536893018566956 19e-35L,
 626         +9.422242548621832065692116736394064 88e-35L,
 627         -3.967500825398862309167306130216418 28e-35L,
 432 -3.967500825398862309167306130216418 28e-35L,
 628         +7.143528991563300614523273615092767 24e-35L,
 629         +1.159871252867985124246517834100444 33e-35L,
 630         +4.696933478358115495309739213201874 47e-35L,
 631         -3.386513175995004710799241984999819 17e-35L,
 632         -8.587318774298247068868655935103874 45e-35L,
 633         -9.605951548749350503185499362246069 09e-35L,
 436 -3.386513175995004710799241984999819 17e-35L,
 437 -8.587318774298247068868655935103874 45e-35L,
 438 -9.605951548749350503185499362246069 09e-35L,
 634         +9.609733932128012784507558697141785 81e-35L,
 635         +6.378397921440028439244761449780848 55e-35L,
 636         +7.792430785695864249456461125169277 70e-35L,
 637         +7.361337767588456524131930836633932 20e-35L,
 638         -6.472995147913347230035214575612170 53e-35L,
 443 -6.472995147913347230035214575612170 53e-35L,
 639         +8.587474417953698694278798062295229 62e-35L,
 640         +2.371815422825174835691651228302690 98e-35L,
 641         -3.026891682096118773004597373421900 31e-37L,
 446 -3.026891682096118773004597373421900 31e-37L,
 642 #endif
 643 };
 449 /* INDENT ON */

 451 /* INDENT OFF */
 646 /*
 647  * return tgamma(x) scaled by 2**-m for 8<x<=171.62... using Stirling's formula
 648  *     log(G(x)) ~= (x-.5)*(log(x)-1) + .5(log(2*pi)-1) + (1/x)*P(1/(x*x))
 649  *                = L1 + L2 + L3,
 650  */
 457 /* INDENT ON */
 651 static struct LDouble
 652 large_gam(long double x, int *m)
 653 {
 459 large_gam(long double x, int *m) {
 654         long double z, t1, t2, t3, z2, t5, w, y, u, r, v;
 655         long double t24 = 16777216.0L, p24 = 1.0L / 16777216.0L;
 656         int n2, j2, k, ix, j, i;
 657         struct LDouble zz;
 658         long double u2, ss_h, ss_l, r_h, w_h, w_l, t4;

 660 /* BEGIN CSTYLED */
 466 /* INDENT OFF */
 661 /*
```

```
662  * compute ss = ss.h+ss.l = log(x)-1 (see tgamma_log.h for details)
663  *
664  *   log(x) - 1 = T1(n) + T2(j) + T3(s), where x = 2**n * y,  1<=y<2,
665  *   j=[64*y], z[j]=1+j/64+1/128, s = (y-z[j])/(y+z[j]), and
666  *        T1(n) = T1[2n,2n+1] = n*log(2)-1,
667  *        T2(j) = T2[2j,2j+1] = log(z[j]),
668  *        T3(s) = 2s + T3[0]s^3 + T3[1]s^5 + ... + T3[6]s^15
669  *   Note
670  *   (1) the leading entries are truncated to 24 binary point.
671  *   (2) Remez error for T3(s) is bounded by 2**(-72.4)
672  *                                    2**(-24)
673  *                                 _________V_________________
674  *               T1(n):     |_________|___________________|
675  *                            _______ _____________________
676  *               T2(j):       |_______|_____________________|
677  *                              ____ _____________________
678  *               2s:            |____|_____________________|
679  *                                   _________________________
680  *          +    T3(s)-2s:           |_________________________|
681  *                        ----------------------------------------
682  *                           [leading] + [Trailing]
683  */
684         /* END CSTYLED */
490         /* INDENT ON */
685         ix = H0_WORD(x);
686         n2 = (ix >> 16) - 0x3fff; /* exponent of x, range:3-10 */
687         y = scalbnl(x, -n2);      /* y = scale x to [1,2] */
688         n2 += n2;                 /* 2n */
689         j = (ix >> 10) & 0x3f;    /* j */
690         z = 1.0078125L + (long double)j * 0.015625L;    /* z[j]=1+j/64+1/128 */
496         z = 1.0078125L + (long double) j * 0.015625L;   /* z[j]=1+j/64+1/128 */
691         j2 = j + j;
692         t1 = y + z;
693         t2 = y - z;
694         r = one / t1;
695         u = r * t2;                       /* u = (y-z)/(y+z) */
696         t1 = CHOPPED(t1);
697         t4 = T2[j2 + 1] + T1[n2 + 1];
698         z2 = u * u;
699         k = H0_WORD(u) & 0x7fffffff;
700         t3 = T2[j2] + T1[n2];

702 #endif /* ! codereview */
703         for (t5 = T3[6], i = 5; i >= 0; i--)
704                 t5 = z2 * t5 + T3[i];

706 #endif /* ! codereview */
707         if ((k >> 16) < 0x3fec) {       /* |u|<2**-19 */
708                 t2 = t4 + u * (two + z2 * t5);
709         } else {
710                 t5 = t4 + (u * z2) * t5;
711                 u2 = u + u;
712                 v = (long double)((int)(u2 * t24)) * p24;
507                 v = (long double) ((int) (u2 * t24)) * p24;
713                 t2 = t5 + r * ((two * t2 - v * t1) - v * (y - (t1 - z)));
714                 t3 += v;
715         }

717 #endif /* ! codereview */
718         ss_h = CHOPPED((t2 + t3));
719         ss_l = t2 - (ss_h - t3);

511 /* INDENT OFF */
721 /*
722  * compute ww = (x-.5)*(log(x)-1) + .5*(log(2pi)-1) + 1/x*(P(1/x^2)))
723  * where ss = log(x) - 1 in already in extra precision
```



```
724  */
516         /* INDENT ON */
725         z = one / x;
726         r = x - half;
727         r_h = CHOPPED((r));
728         w_h = r_h * ss_h + hln2pim1_h;
729         z2 = z * z;
730         w = (r - r_h) * ss_h + r * ss_l;
731         t1 = GP[19];

733 #endif /* ! codereview */
734         for (i = 18; i > 0; i--)
735                 t1 = z2 * t1 + GP[i];

737 #endif /* ! codereview */
738         w += hln2pim1_l;
739         w_l = z * (GP[0] + z2 * t1) + w;
740         k = (int)((w_h + w_l) * invln2_32 + half);
524         k = (int) ((w_h + w_l) * invln2_32 + half);

742         /* compute the exponential of w_h+w_l */

744         j = k & 0x1f;
745         *m = k >> 5;
746         t3 = (long double)k;
530         t3 = (long double) k;

748         /* perform w - k*ln2_32 (represent as w_h - w_l) */
749         t1 = w_h - t3 * ln2_32hi;
750         t2 = t3 * ln2_32lo;
751         w = t2 - w_l;
752         w_h = t1 - w;
753         w_l = w - (t1 - w_h);

755         /* compute exp(w_h-w_l) */
756         z = w_h - w_l;

758 #endif /* ! codereview */
759         for (t1 = Et[10], i = 9; i >= 0; i--)
760                 t1 = z * t1 + Et[i];

762 #endif /* ! codereview */
763         t3 = w_h - (w_l - (z * z) * t1);        /* t3 = expm1(z) */
764         zz.l = S_trail[j] * (one + t3) + S[j] * t3;
765         zz.h = S[j];
766         return (zz);
767 }


541 /* INDENT OFF */
770 /*
771  * kpsin(x)= sin(pi*x)/pi
772  *                3        5        7        9        11                27
773  *      = x+ks[0]*x +ks[1]*x +ks[2]*x +ks[3]*x +ks[4]*x  + ... + ks[12]*x
774  */
775 static const long double ks[] = {
776         -1.6449340668482264364724151666460251870515890287 0e+0000L,
777         +8.1174242528335364363700277240587423809499572616 0e-0001L,
778         -1.9075182412208421369647211183533736623228272393 3e-0001L,
779         +2.6147847817654800504653261356324128811539551708 4e-0002L,
780         -2.3460810354558236375089307264711782944801647997 1e-0003L,
781         +1.4842879303107100368460664721253402755626204015 8e-0004L,
782         -6.9758736616563804651846272225276812261595289869 8e-0006L,
783         +2.5312174041370253692865927174718750093484005792 9e-0007L,
784         -7.3047118222138599039768364169576612130193362195 6e-0009L,
785         +1.7165384745116349573995824969554931398797358988 4e-0010L,
```

```
        -3.348133147145607761222457969290548134583414205 65e-0012L,
        +5.507249922626220334494878083069691354314117530 47e-0014L,
        -7.676781327535779986012343932158022211042369799 28e-0016L,
};
/* INDENT ON */

/*
 * assume x is not tiny and positive
 */
static struct LDouble
kpsin(long double x)
{
kpsin(long double x) {
        long double z, t1, t2;
        struct LDouble xx;
        int i;

        z = x * x;
        xx.h = x;

#endif /* ! codereview */
        for (t2 = ks[12], i = 11; i > 0; i--)
                t2 = z * t2 + ks[i];

#endif /* ! codereview */
        t1 = z * x;
        t2 *= z * t1;
        xx.l = t1 * ks[0] + t2;
        return (xx);
}


/* INDENT OFF */
/*
 * kpcos(x)= cos(pi*x)/pi
 *                    2        4        6        8        10        12
 *      = 1/pi +kc[0]*x +kc[1]*x +kc[2]*x +kc[3]*x +kc[4]*x  +kc[5]*x
 *
 *                    2        4        6        8        10          22
 *      = 1/pi - pi/2*x +kc[0]*x +kc[1]*x +kc[2]*x +kc[3]*x  +...+kc[9]*x
 *
 * -pi/2*x*x = (npi_2_h + npi_2_l) * (x_f+x_l)*(x_f+x_l)
 *         =  npi_2_h*(x_f+x_l)*(x_f+x_l) + npi_2_l*x*x
 *         =  npi_2_h*x_f*x_f + npi_2_h*(x*x-x_f*x_f) + npi_2_l*x*x
 *         =  npi_2_h*x_f*x_f + npi_2_h*(x+x_f)*(x-x_f) + npi_2_l*x*x
 * Here x_f = (long double) (float)x
 * Note that pi/2(in hex) =
 *  1.921FB54442D18469898CC51701B839A252049C1114CF98E804177D4C76273644A29
 * npi_2_h = -pi/2 chopped to 25 bits = -1.921FB5000000000000000000000000 =
 *  -1.5707963109016418457031250000000000 and
 * npi_2_l =
 *  -0.0000004442D18469898CC51701B839A252049C1114CF98E804177D4C76273644A29 =
 *  -.0000000158932547735281966916397514420985846996875529104874722961539 =
 *  -1.589325477352819669163975144209858469968755291048747229615 3e-8
 * 1/pi(in hex) =
 *  .517CC1B727220A94FE13ABE8FA9A6EE06DB14ACC9E21C820FF28B1D5EF5DE2B
 * will be splitted into:
 *  one_pi_h = 1/pi chopped to 48 bits = .517CC1B727220000000000...  and
 *  one_pi_l = .0000000000000A94FE13ABE8FA9A6EE06DB14ACC9E21C820FF28B1D5EF5DE2B
 */

static const long double
#if defined(__x86)
        one_pi_h = 0.318309886148199439048767089843750 0L,       /* 31 bits */
        one_pi_l = 3.559123248900043690127872406891929148e-11L,
one_pi_h = 0.318309886148199439048767089843750 0L,       /* 31 bits */
```



```
one_pi_l = 3.559123248900043690127872406891929148e-11L,
#else
        one_pi_h = 0.318309886183790524682990508154034614562988281 25L,
        one_pi_l = 1.468547770185909941095059310102309128974953346881 17e-16L,
one_pi_h = 0.318309886183790524682990508154034614562988281 25L,
one_pi_l = 1.468547770185909941095059310102309128974953346881 17e-16L,
#endif
        npi_2_h = -1.570796310901641845703125000000000L,
        npi_2_l = -1.589325477352819669163975144209858469968755291 0e-8L;
npi_2_h = -1.570796310901641845703125000000000L,
npi_2_l = -1.589325477352819669163975144209858469968755291 0e-8L;

static const long double kc[] = {
        +1.291928195012492507311513127795489187653207284 89e+0000L,
        -4.250273399795575739760295969293192070094440903 66e-0001L,
        +7.490806616509900961096729546183176238884216286 13e-0002L,
        -8.214588661112822879855394641739765554360502151 20e-0003L,
        +6.142025788095292285032052551657612047502116034 02e-0004L,
        -3.330734326911496070072173303025952671795459087 40e-0005L,
        +1.369709590478320857968097454615308655979936802 04e-0006L,
        -4.417807742625835144502465127272018062172710973 36e-0008L,
        +1.147414092123818588200165676644881234786607057 59e-0009L,
        -2.442612361147073745584375006543810063005027496 32e-0011L,
};
/* INDENT ON */

/*
 * assume x is not tiny and positive
 */
static struct LDouble
kpcos(long double x)
{
kpcos(long double x) {
        long double z, t1, t2, t3, t4, x4, x8;
        int i;
        struct LDouble xx;

        z = x * x;
        xx.h = one_pi_h;
        t1 = (long double)((float)x);
        t1 = (long double) ((float) x);
        x4 = z * z;
        t2 = npi_2_l * z + npi_2_h * (x + t1) * (x - t1);

#endif /* ! codereview */
        for (i = 8, t3 = kc[9]; i >= 0; i--)
                t3 = z * t3 + kc[i];

#endif /* ! codereview */
        t3 = one_pi_l + x4 * t3;
        t4 = t1 * t1 * npi_2_h;
        t2 = t2 + t3;
        xx.l = x8 + t4;
        return (xx);
}

/* INDENT OFF */
static const long double
/* 0.134861805732790769689793935774652917006425111395524293982 33 */
        /* 0.134861805732790769689793935774652917006425111395524293982 33 */
#if defined(__x86)
        t0z1 = 0.134861805732790769677938505499703580881 0L,
        t0z1_l = 1.185543027494933612539271715025737961465 4e-20L,
t0z1   =  0.134861805732790769677938505499703580881 0L,
t0z1_l =  1.185543027494933612539271715025737961465 4e-20L,
#else
```

```
901         t0z1 = 0.1348618057327907696897939357746529168654L,
902         t0z1_l = 1.4102088588676879418739164486159514674310e-37L,
650 t0z1   =  0.1348618057327907696897939357746529168654L,
651 t0z1_l =  1.4102088588676879418739164486159514674310e-37L,
903 #endif
904 /* 0.4616321449683623412626595423257213284681962040064463512959 9 */
653         /* 0.4616321449683623412626595423257213284681962040064463512959 9 */
905 #if defined(__x86)
906         t0z2 = 0.4616321449683623412538115843295472018326L,
907         t0z2_l = 8.8479579961741266355853230503926174703064 0e-21L,
655 t0z2   =  0.4616321449683623412538115843295472018326L,
656 t0z2_l =  8.8479579961741266355853230503926174703064 0e-21L,
908 #else
909         t0z2 = 0.4616321449683623412626595423257213234331 8L,
910         t0z2_l = 5.0350116232961638046530266648091627161110 1e-36L,
658 t0z2   =  0.4616321449683623412626595423257213234331 8L,
659 t0z2_l =  5.0350116232961638046530266648091627161110 1e-36L,
911 #endif
912 /* 0.8197731011005006017878687049216069963117440784624517911958 6 */
661         /* 0.8197731011005006017878687049216069963117440784624517911958 6 */
913 #if defined(__x86)
914         t0z3 = 0.8197731011005006017877336232935192583681 7L,
915         t0z3_l = 1.3508162808773794356580770525345745562562 30e-22L
663 t0z3   =  0.8197731011005006017877336232935192583681 7L,
664 t0z3_l =  1.3508162808773794356580770525345745562562 30e-22L
916 #else
917         t0z3 = 0.8197731011005006017878687049216069516957449L,
918         t0z3_l = 4.4615999169470144190454926159335516488573 80e-35L
666 t0z3   =  0.8197731011005006017878687049216069516957449L,
667 t0z3_l =  4.4615999169470144190454926159335516488573 80e-35L
919 #endif
920 ;
670 /* INDENT ON */

922 /*
923  * gamma(x+i) for 0 <= x < 1
924  */
925 static struct LDouble
926 gam_n(int i, long double x)
927 {
928         struct LDouble rr = { 0.0L, 0.0L }, yy;
676 gam_n(int i, long double x) {
677         struct LDouble rr = {0.0L, 0.0L}, yy;
929         long double r1, r2, t2, z, xh, xl, yh, yl, zh, z1, z2, zl, x5, wh, wl;

931         /* compute yy = gamma(x+1) */
932         if (x > 0.2845L) {
933                 if (x > 0.6374L) {
934                         r1 = x - t0z3;
935                         r2 = CHOPPED((r1 - t0z3_l));
936                         t2 = r1 - r2;
937                         yy = GT3(r2, t2 - t0z3_l);
938                 } else {
939                         r1 = x - t0z2;
940                         r2 = CHOPPED((r1 - t0z2_l));
941                         t2 = r1 - r2;
942                         yy = GT2(r2, t2 - t0z2_l);
943                 }
944         } else {
945                 r1 = x - t0z1;
946                 r2 = CHOPPED((r1 - t0z1_l));
947                 t2 = r1 - r2;
948                 yy = GT1(r2, t2 - t0z1_l);
949         }

951 #endif /* ! codereview */
```



```
952         /* compute gamma(x+i) = (x+i-1)*...*(x+1)*yy, 0<i<8 */
953         switch (i) {
954         case 0:                         /* yy/x */
955                 r1 = one / x;
956                 xh = CHOPPED((x));      /* x is not tiny */
957                 rr.h = CHOPPED(((yy.h + yy.l) * r1));
958                 rr.l = r1 * (yy.h - rr.h * xh) - ((r1 * rr.h) * (x - xh) - r1 *
959                     yy.l);
699                 rr.l = r1 * (yy.h - rr.h * xh) - ((r1 * rr.h) * (x - xh) -
700                         r1 * yy.l);
960                 break;
961         case 1:                         /* yy */
962                 rr.h = yy.h;
963                 rr.l = yy.l;
964                 break;
965         case 2:                         /* (x+1)*yy */
966                 z = x + one;            /* may not be exact */
967                 zh = CHOPPED((z));
968                 rr.h = zh * yy.h;
969                 rr.l = z * yy.l + (x - (zh - one)) * yy.h;
970                 break;
971         case 3:                         /* (x+2)*(x+1)*yy */
972                 z1 = x + one;
973                 z2 = x + 2.0L;
974                 z = z1 * z2;
975                 xh = CHOPPED((z));
976                 zh = CHOPPED((z1));
977                 xl = (x - (zh - one)) * (z2 + zh) - (xh - zh * (zh + one));

979                 rr.h = xh * yy.h;
980                 rr.l = z * yy.l + xl * yy.h;
981                 break;

983         case 4:                         /* (x+1)*(x+3)*(x+2)*yy */
984                 z1 = x + 2.0L;
985                 z2 = (x + one) * (x + 3.0L);
986                 zh = CHOPPED(z1);
987                 zl = x - (zh - 2.0L);
988                 xh = CHOPPED(z2);
989                 xl = zl * (zh + z1) - (xh - (zh * zh - one));

991                 /* wh+wl=(x+2)*yy */
992                 wh = CHOPPED((z1 * (yy.h + yy.l)));
993                 wl = (zl * yy.h + z1 * yy.l) - (wh - zh * yy.h);

995                 rr.h = xh * wh;
996                 rr.l = z2 * wl + xl * wh;

998                 break;
999         case 5:                         /* ((x+1)*(x+4)*(x+2)*(x+3))*yy */
1000                z1 = x + 2.0L;
1001                z2 = x + 3.0L;
1002                z = z1 * z2;
1003                zh = CHOPPED((z1));
1004                yh = CHOPPED((z));
1005                yl = (x - (zh - 2.0L)) * (z2 + zh) - (yh - zh * (zh + one));
1006                z2 = z - 2.0L;
1007                z *= z2;
1008                xh = CHOPPED((z));
1009                xl = yl * (z2 + yh) - (xh - yh * (yh - 2.0L));
1010                rr.h = xh * yy.h;
1011                rr.l = z * yy.l + xl * yy.h;
1012                break;
1013        case 6:                         /* ((x+1)*(x+2)*(x+3)*(x+4)*(x+5))*yy */
1014                z1 = x + 2.0L;
1015                z2 = x + 3.0L;
```

```
                z = z1 * z2;
                zh = CHOPPED((z1));
                yh = CHOPPED((z));
                z1 = x - (zh - 2.0L);
                yl = z1 * (z2 + zh) - (yh - zh * (zh + one));
                z2 = z - 2.0L;
                x5 = x + 5.0L;
                z *= z2;
                xh = CHOPPED(z);
                zh += 3.0;
                xl = yl * (z2 + yh) - (xh - yh * (yh - 2.0L));

                /*
                 * xh+xl=(x+1)*...*(x+4)
                 * wh+wl=(x+5)*yy
                 */
 768                                            /* xh+xl=(x+1)*...*(x+4) */
 769            /* wh+wl=(x+5)*yy */
                wh = CHOPPED((x5 * (yy.h + yy.l)));
                wl = (z1 * yy.h + x5 * yy.l) - (wh - zh * yy.h);
                rr.h = wh * xh;
                rr.l = z * wl + xl * wh;
                break;
        case 7:                 /* ((x+1)*(x+2)*(x+3)*(x+4)*(x+5)*(x+6))*yy */
                z1 = x + 3.0L;
                z2 = x + 4.0L;
                z = z2 * z1;
                zh = CHOPPED((z1));
                yh = CHOPPED((z));      /* yh+yl = (x+3)(x+4) */
                yl = (x - (zh - 3.0L)) * (z2 + zh) - (yh - (zh * (zh + one)));
                z1 = x + 6.0L;
                z2 = z - 2.0L;          /* z2 = (x+2)*(x+5) */
                z *= z2;
                xh = CHOPPED((z));
                xl = yl * (z2 + yh) - (xh - yh * (yh - 2.0L));

                /*
                 * xh+xl=(x+2)*...*(x+5)
                 * wh+wl=(x+1)(x+6)*yy
                 */
 787                                            /* xh+xl=(x+2)*...*(x+5) */
 788            /* wh+wl=(x+1)(x+6)*yy */
                z2 -= 4.0L;             /* z2 = (x+1)(x+6) */
                wh = CHOPPED((z2 * (yy.h + yy.l)));
                wl = (z2 * yy.l + yl * yy.h) - (wh - (yh - 6.0L) * yy.h);
                rr.h = wh * xh;
                rr.l = z * wl + xl * wh;
        }

#endif /* ! codereview */
        return (rr);
}

long double
tgammal(long double x)
{
 795 tgammal(long double x) {
        struct LDouble ss, ww;
        long double t, t1, t2, t3, t4, t5, w, y, z, z1, z2, z3, z5;
        int i, j, m, ix, hx, xk;
        unsigned lx;

        hx = H0_WORD(x);
        lx = H3_WORD(x);
        ix = hx & 0x7fffffff;
        y = x;
```



```
        if (ix < 0x3f8e0000)            /* x < 2**-113 */
 805    if (ix < 0x3f8e0000) {  /* x < 2**-113 */
                return (one / x);

 807    }
        if (ix >= 0x7fff0000)
                return (x * ((hx < 0) ? zero : x));     /* Inf or NaN */

 809            return (x * ((hx < 0)? zero : x));      /* Inf or NaN */
        if (x > overflow)                               /* overflow threshold */
                return (x * 1.0e4932L);

#endif /* ! codereview */
        if (hx >= 0x40020000) {                         /* x >= 8 */
                ww = large_gam(x, &m);
                w = ww.h + ww.l;
                return (scalbnl(w, m));
        }

        if (hx > 0) {                   /* 0 < x < 8 */
                i = (int)x;
                ww = gam_n(i, x - (long double)i);
 812            i = (int) x;
 813            ww = gam_n(i, x - (long double) i);
                return (ww.h + ww.l);
        }

        /*
         * negative x
         */

 816    /* INDENT OFF */
 817    /* negative x */
        /*
         * compute xk =
         *      -2 ... x is an even int (-inf is considered an even #)
         *      -1 ... x is an odd int
         *      +0 ... x is not an int but chopped to an even int
         *      +1 ... x is not an int but chopped to an odd int
         */
 825    /* INDENT ON */
        xk = 0;
#if defined(__x86)
        if (ix >= 0x403e0000) {         /* x >= 2**63 } */
                if (ix >= 0x403f0000)
                        xk = -2;
                else
                        xk = -2 + (lx & 1);
#else
        if (ix >= 0x406f0000) {         /* x >= 2**112 */
                if (ix >= 0x40700000)
                        xk = -2;
                else
                        xk = -2 + (lx & 1);
#endif
        } else if (ix >= 0x3fff0000) {
                w = -x;
                t1 = floorl(w);
                t2 = t1 * half;
                t3 = floorl(t2);

#endif /* ! codereview */
                if (t1 == w) {
                        if (t2 == t3)
                                xk = -2;
```

```
                        else
                                xk = -1;
                } else {
                        if (t2 == t3)
                                xk = 0;
                        else
                                xk = 1;
                }
        }

        if (xk < 0) {
                /* return NaN. Ideally gamma(-n)= (-1)**(n+1) * inf */
                return ((x - x) / (x - x));
                return (x - x) / (x - x);
        }

        /*
         * negative underflow thresold -(1774+9ulp)
         */
        if (x < -1774.00000000000000000000000000017749370L) {
                z = tiny / x;

#endif /* ! codereview */
                if (xk == 1)
                        z = -z;

#endif /* ! codereview */
                return (z * tiny);
        }


        /* INDENT OFF */
        /*
         * now compute gamma(x) by  -1/((sin(pi*y)/pi)*gamma(1+y)), y = -x
         */

#endif /* ! codereview */
        /*
         * First compute ss = -sin(pi*y)/pi so that
         * gamma(x) = 1/(ss*gamma(1+y))
         */
        /* INDENT ON */
        y = -x;
        j = (int)y;
        z = y - (long double)j;

        if (z > 0.31830988618379067153776 75L) {
        j = (int) y;
        z = y - (long double) j;
        if (z > 0.31830988618379067153776 75L)
                if (z > 0.68169011381620932846223 25L)
                        ss = kpsin(one - z);
                else
                        ss = kpcos(0.5L - z);
        } else {
        else
                ss = kpsin(z);
        }

#endif /* ! codereview */
        if (xk == 0) {
                ss.h = -ss.h;
                ss.l = -ss.l;
        }

        /* Then compute ww = gamma(1+y), note that result scale to 2**m */
```



```
        m = 0;

#endif /* ! codereview */
        if (j < 7) {
                ww = gam_n(j + 1, z);
        } else {
                w = y + one;

#endif /* ! codereview */
                if ((lx & 1) == 0) {    /* y+1 exact (note that y<184) */
                        ww = large_gam(w, &m);
                } else {
                        t = w - one;

#endif /* ! codereview */
                        if (t == y) {   /* y+one exact */
                                ww = large_gam(w, &m);
                        } else {        /* use y*gamma(y) */
                                if (j == 7)
                                        ww = gam_n(j, z);
                                else
                                        ww = large_gam(y, &m);

#endif /* ! codereview */
                                t4 = ww.h + ww.l;
                                t1 = CHOPPED((y));
                                t2 = CHOPPED((t4));
                                /* t4 will not be too large */
                                ww.l = y * (ww.l - (t2 - ww.h)) + (y - t1) * t2;
                                ww.h = t1 * t2;
                        }
                }
        }

        /* compute 1/(ss*ww) */
        t3 = ss.h + ss.l;
        t4 = ww.h + ww.l;
        t1 = CHOPPED((t3));
        t2 = CHOPPED((t4));
        z1 = ss.l - (t1 - ss.h);        /* (t1,z1) = ss */
        z2 = ww.l - (t2 - ww.h);        /* (t2,z2) = ww */
        t3 = t3 * t4;                   /* t3 = ss*ww */
        z3 = one / t3;                  /* z3 = 1/(ss*ww) */
        t5 = t1 * t2;
        z5 = z1 * t4 + t1 * z2;         /* (t5,z5) = ss*ww */
        t1 = CHOPPED((t3));             /* (t1,z1) = ss*ww */
        z1 = z5 - (t1 - t5);
        t2 = CHOPPED((z3));             /* leading 1/(ss*ww) */
        z2 = z3 * (t2 * z1 - (one - t2 * t1));
        z = t2 - z2;

        return (scalbnl(z, -m));
}
```

```
**********************************************************
    1786 Thu Jun  6 20:40:39 2019
new/usr/src/lib/libm/common/m9x/trunc.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __trunc = trunc

#include "libm.h"

double
trunc(double x)
{
trunc(double x) {
        union {
                unsigned i[2];
                double d;
        } xx;

#endif /* ! codereview */
        unsigned hx, sx, i;

        xx.d = x;
        hx = xx.i[HIWORD] & ~0x80000000;
        sx = xx.i[HIWORD] & 0x80000000;

#endif /* ! codereview */
        if (hx < 0x43300000) {          /* |x| < 2^52 */
                if (hx < 0x3ff00000)    /* |x| < 1 */
                        return (sx ? -0.0 : 0.0);

                /* chop x at the integer bit */
                if (hx < 0x41300000) {
                        i = 1 << (0x412 - (hx >> 20));
                        xx.i[HIWORD] &= ~(i | (i - 1));
                        xx.i[LOWORD] = 0;
```



```
                } else {
                        i = 1 << (0x432 - (hx >> 20));
                        xx.i[LOWORD] &= ~(i | (i - 1));
                }

#endif /* ! codereview */
                return (xx.d);
        } else if (hx < 0x7ff00000) {
        } else if (hx < 0x7ff00000)
                return (x);
        } else {
        else
#if defined(FPADD_TRAPS_INCOMPLETE_ON_NAN)
                return (hx >= 0x7ff80000 ? x : x + x);

#endif /* ! codereview */
        /* assumes sparc-like QNaN */
#else
                return (x + x);
#endif
        }
#endif /* ! codereview */
}
```

```
**********************************************************
    1621 Thu Jun  6 20:40:40 2019
new/usr/src/lib/libm/common/m9x/truncf.c
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __truncf = truncf

#include "libm.h"

float
truncf(float x)
{
truncf(float x) {
        union {
                unsigned i;
                float f;
        } xx;

#endif /* ! codereview */
        unsigned hx, sx, i;

        xx.f = x;
        hx = xx.i & ~0x80000000;
        sx = xx.i & 0x80000000;

#endif /* ! codereview */
        if (hx < 0x4b000000) {          /* |x| < 2^23 */
                if (hx < 0x3f800000)    /* |x| < 1 */
                        return (sx ? -0.0F : 0.0F);

                /* chop x at the integer bit */
                i = 1 << (0x95 - (hx >> 23));
                xx.i &= ~((i << 1) - 1);
                return (xx.f);
        } else if (hx < 0x7f800000) {   /* |x| is integral */
```



```
        } else if (hx < 0x7f800000)     /* |x| is integral */
                return (x);
        } else {
        else
#if defined(FPADD_TRAPS_INCOMPLETE_ON_NAN)
                return (hx > 0x7f800000 ? x * x : x + x);
#else
                return (x + x);
#endif
        }
#endif /* ! codereview */
}
```

```
**********************************************************
    2685 Thu Jun  6 20:40:40 2019
new/usr/src/lib/libm/common/m9x/truncl.c
11210 libm should be cstyle(10NBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */

#endif /* ! codereview */
/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#pragma weak __truncl = truncl

#include "libm.h"

#if defined(__sparc)
long double
truncl(long double x)
{
truncl(long double x) {
        union {
                unsigned i[4];
                long double q;
        } xx;

#endif /* ! codereview */
        unsigned hx, sx;
        int j;

        xx.q = x;
        sx = xx.i[0] & 0x80000000;
        hx = xx.i[0] & ~0x80000000;

        /* handle trivial cases */
        if (hx >= 0x406f0000)           /* |x| >= 2^112 + ... or x is nan */
                return (hx >= 0x7fff0000 ? x + x : x);

        /* handle |x| < 1 */
        if (hx < 0x3fff0000)
                return (sx ? -0.0L : 0.0L);
```



```
        j = 0x406f - (hx >> 16);        /* 1 <= j <= 112 */
        xx.i[0] = hx;

#endif /* ! codereview */
        if (j >= 96) {                  /* 96 <= j <= 112 */
                xx.i[0] &= ~((1 << (j - 96)) - 1);
                xx.i[1] = xx.i[2] = xx.i[3] = 0;
        } else if (j >= 64) {           /* 64 <= j <= 95 */
                xx.i[1] &= ~((1 << (j - 64)) - 1);
                xx.i[2] = xx.i[3] = 0;
        } else if (j >= 32) {           /* 32 <= j <= 63 */
                xx.i[2] &= ~((1 << (j - 32)) - 1);
                xx.i[3] = 0;
        } else {                        /* 1 <= j <= 31 */
        } else                                  /* 1 <= j <= 31 */
                xx.i[3] &= ~((1 << j) - 1);
        }
#endif /* ! codereview */

        /* negate result if need be */
        if (sx)
                xx.i[0] |= 0x80000000;

#endif /* ! codereview */
        return (xx.q);
}
#elif defined(__x86)
long double
truncl(long double x)
{
truncl(long double x) {
        union {
                unsigned i[3];
                long double e;
        } xx;

#endif /* ! codereview */
        int ex, sx, i;

        xx.e = x;
        ex = xx.i[2] & 0x7fff;
        sx = xx.i[2] & 0x8000;

#endif /* ! codereview */
        if (ex < 0x403e) {              /* |x| < 2^63 */
                if (ex < 0x3fff)        /* |x| < 1 */
                        return (sx ? -0.0L : 0.0L);

                /* chop x at the integer bit */
                if (ex < 0x401e) {
                        i = 1 << (0x401d - ex);
                        xx.i[1] &= ~(i | (i - 1));
                        xx.i[0] = 0;
                } else {
                        i = 1 << (0x403d - ex);
                        xx.i[0] &= ~(i | (i - 1));
                }

#endif /* ! codereview */
                return (xx.e);
        } else if (ex < 0x7fff) {       /* x is integral */
        } else if (ex < 0x7fff) /* x is integral */
                return (x);
        } else {                        /* inf or nan */
        else                    /* inf or nan */
                return (x + x);
```

```
        }
#endif /* ! codereview */
}
#else
#error Unknown architecture
#endif /* defined(__sparc) || defined(__x86) */
```

```
**********************************************************
    7088 Thu Jun  6 20:40:41 2019
new/usr/src/lib/libm/i386/src/libm_inlines.h
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

/*
 * Copyright 2011, Richard Lowe
 */

/* Functions in this file are duplicated in locallibm.il.  Keep them in sync */

#ifndef _LIBM_INLINES_H
#define _LIBM_INLINES_H

#ifdef __GNUC__

#ifdef __cplusplus
extern "C" {
#endif

#include <sys/types.h>
#include <sys/ieeefp.h>

#define _LO_WORD(x)     ((uint32_t *)&x)[0]
#define _HI_WORD(x)     ((uint32_t *)&x)[1]
#define _HIER_WORD(x)   ((uint32_t *)&x)[2]

extern __GNU_INLINE double
__inline_sqrt(double a)
{
        double ret;

        __asm__ __volatile__("fsqrt\n\t" : "=t" (ret) : "0" (a) : "cc");
        return (ret);
}
______unchanged_portion_omitted_

extern __GNU_INLINE double
rint(double a)
{
```



```
rint(double a) {
        return (__inline_rint(a));
}
______unchanged_portion_omitted_
#endif

#endif  /* __GNUC__ */

#endif /* _LIBM_INLINES_H */
```

```
**********************************************************
    7050 Thu Jun  6 20:40:41 2019
new/usr/src/lib/libm/sparc/src/libm_inlines.h
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

/*
 * Copyright 2011, Richard Lowe.
 */

/* Functions in this file are duplicated in locallibm.il.  Keep them in sync */

#ifndef _LIBM_INLINES_H
#define _LIBM_INLINES_H

#ifdef __GNUC__

#include <sys/types.h>
#include <sys/ieeefp.h>

#ifdef __cplusplus
extern "C" {
#endif

extern __GNU_INLINE double
__inline_sqrt(double d)
{
        double ret;

        __asm__ __volatile__("fsqrtd %1,%0\n\t" : "=e" (ret) : "e" (d));
        return (ret);
}
______unchanged_portion_omitted_
#endif

#endif  /* __GNUC */

#endif /* _LIBM_INLINES_H */
```

```
**********************************************************
    6496 Thu Jun  6 20:40:42 2019
new/usr/src/lib/libm/sparcv9/src/libm_inlines.h
11210 libm should be cstyle(1ONBLD) clean
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

/*
 * Copyright 2011, Richard Lowe.
 */

/* Functions in this file are duplicated in locallibm.il.  Keep them in sync */

#ifndef _LIBM_INLINES_H
#define _LIBM_INLINES_H

#ifdef __GNUC__

#include <sys/types.h>
#include <sys/ieeefp.h>

#ifdef __cplusplus
extern "C" {
#endif

extern __GNU_INLINE enum fp_class_type
fp_classf(float f)
{
        enum fp_class_type ret;
        int fint;               /* scratch for f as int */
        uint64_t tmp;

        __asm__ __volatile__(
            "fabss  %3,%3\n\t"
            "st     %3,%1\n\t"
            "ld     %1,%0\n\t"
            "orcc   %%g0,%0,%%g0\n\t"
            "be,pn  %%icc,2f\n\t"
            "nop\n\t"
            "1:\n\t"
            "sethi  %%hi(0x7f800000),%2\n\t"
            "andcc  %0,%2,%%g0\n\t"
```



```
            "bne,pt %%icc,1f\n\t"
            "nop\n\t"
            "or     %%g0,1,%0\n\t"
            "ba     2f\n\t"     /* subnormal */
            "nop\n\t"
            "1:\n\t"
            "subcc  %0,%2,%%g0\n\t"
            "bge,pn %%icc,1f\n\t"
            "nop\n\t"
            "or     %%g0,2,%0\n\t"
            "ba     2f\n\t"     /* normal */
            "nop\n\t"
            "1:\n\t"
            "bg,pn  %%icc,1f\n\t"
            "nop\n\t"
            "or     %%g0,3,%0\n\t"
            "ba     2f\n\t"     /* infinity */
            "nop\n\t"
            "1:\n\t"
            "sethi  %%hi(0x00400000),%2\n\t"
            "andcc  %0,%2,%%g0\n\t"
            "or     %%g0,4,%0\n\t"
            "bne,pt %%icc,2f\n\t" /* quiet NaN */
            "nop\n\t"
            "or     %%g0,5,%0\n\t" /* signalling NaN */
            "2:\n\t"
            : "=r" (ret), "=m" (fint), "=r" (tmp), "+f" (f)
            :
            : "cc");

        return (ret);
}
_____unchanged_portion_omitted_
#endif

#endif  /* __GNUC__ */

#endif /* _LIBM_INLINES_H */
```