```
**********************************************************
    4839 Thu Oct  9 19:48:51 2014
new/usr/src/head/complex.h
patching complex.h - https://www.illumos.org/issues/3880
patch01 - 693 import Sun Devpro Math Library
patching complex.h - https://www.illumos.org/issues/3880
patch01 - 693 import Sun Devpro Math Library
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" 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 2004 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#ifndef _COMPLEX_H
#define _COMPLEX_H

#ifdef  __cplusplus
extern "C" {
#endif

/* #if !defined(__cplusplus) */

/*
 * Compilation environments for Solaris must provide the _Imaginary datatype
 * and the compiler intrinsics _Complex_I and _Imaginary_I
 */
#if defined(__SUNPRO_C)
#define _Complex_I      _Complex_I
#define _Imaginary_I    _Imaginary_I
#else
#define _Complex_I      1.0fi
#define _Imaginary_I    1.0fi
#endif
#define complex         _Complex
#define imaginary       _Imaginary
#undef  I
#define I               _Imaginary_I

extern float cabsf(float complex);
extern float cargf(float complex);
extern float cimagf(float complex);
extern float crealf(float complex);
extern float complex cacosf(float complex);
```



```
extern float complex cacoshf(float complex);
extern float complex casinf(float complex);
extern float complex casinhf(float complex);
extern float complex catanf(float complex);
extern float complex catanhf(float complex);
extern float complex ccosf(float complex);
extern float complex ccoshf(float complex);
extern float complex cexpf(float complex);
extern float complex clogf(float complex);
extern float complex conjf(float complex);
extern float complex cpowf(float complex, float complex);
extern float complex cprojf(float complex);
extern float complex csinf(float complex);
extern float complex csinhf(float complex);
extern float complex csqrtf(float complex);
extern float complex ctanf(float complex);
extern float complex ctanhf(float complex);

extern double cabs(double complex);
extern double carg(double complex);
extern double cimag(double complex);
extern double creal(double complex);
extern double complex cacos(double complex);
extern double complex cacosh(double complex);
extern double complex casin(double complex);
extern double complex casinh(double complex);
extern double complex catan(double complex);
extern double complex catanh(double complex);
extern double complex ccos(double complex);
extern double complex ccosh(double complex);
extern double complex cexp(double complex);
#if defined(__PRAGMA_REDEFINE_EXTNAME)
#pragma redefine_extname clog __clog
#else
#undef  clog
#define clog    __clog
#endif
extern double complex clog(double complex);
extern double complex conj(double complex);
extern double complex cpow(double complex, double complex);
extern double complex cproj(double complex);
extern double complex csin(double complex);
extern double complex csinh(double complex);
extern double complex csqrt(double complex);
extern double complex ctan(double complex);
extern double complex ctanh(double complex);

extern long double cabsl(long double complex);
extern long double cargl(long double complex);
extern long double cimagl(long double complex);
extern long double creall(long double complex);
extern long double complex cacoshl(long double complex);
extern long double complex cacosl(long double complex);
extern long double complex casinhl(long double complex);
extern long double complex casinl(long double complex);
extern long double complex catanhl(long double complex);
extern long double complex catanl(long double complex);
extern long double complex ccoshl(long double complex);
extern long double complex ccosl(long double complex);
extern long double complex cexpl(long double complex);
extern long double complex clogl(long double complex);
extern long double complex conjl(long double complex);
extern long double complex cpowl(long double complex, long double complex);
extern long double complex cprojl(long double complex);
extern long double complex csinhl(long double complex);
extern long double complex csinl(long double complex);
```

```
125 extern long double complex csqrtl(long double complex);
126 extern long double complex ctanhl(long double complex);
127 extern long double complex ctanl(long double complex);

129 /* #endif */    /* !defined(__cplusplus) */
130 #ifdef  __cplusplus
131 }
132 #endif

134 #endif  /* _COMPLEX_H */
```

```
**********************************************************
    6821 Thu Oct  9 19:48:51 2014
new/usr/src/head/floatingpoint.h
patching floatingpoint.h - https://www.illumos.org/issues/3853
patch01 - 693 import Sun Devpro Math Library
patching floatingpoint.h - https://www.illumos.org/issues/3853
patch01 - 693 import Sun Devpro Math Library
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*      Copyright (C) 1989 AT&T */
/*        All Rights Reserved */

/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 */
/*
 * Copyright 2004 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

#ifndef _FLOATINGPOINT_H
#define _FLOATINGPOINT_H

#ifdef __STDC__
#include <stdio_tag.h>
#endif
#include <sys/ieeefp.h>

#ifdef __cplusplus
extern "C" {
#endif

/*
 * <floatingpoint.h> contains definitions for constants, types, variables,
 * and functions for:
 *      IEEE floating-point arithmetic base conversion;
 *      IEEE floating-point arithmetic modes;
 *      IEEE floating-point arithmetic exception handling.
 */

#ifndef __P
#ifdef __STDC__
#define __P(p)  p
#else
#define __P(p)  ()
#endif
#endif  /* !defined(__P) */
```



```
#if defined(__STDC__) && !defined(_FILEDEFED)
#define _FILEDEFED
typedef __FILE FILE;
#endif

#define N_IEEE_EXCEPTION 5      /* Number of floating-point exceptions. */

typedef int sigfpe_code_type;   /* Type of SIGFPE code. */

typedef void (*sigfpe_handler_type)();  /* Pointer to exception handler */

#define SIGFPE_DEFAULT (void (*)())0    /* default exception handling */
#define SIGFPE_IGNORE  (void (*)())1    /* ignore this exception or code */
#define SIGFPE_ABORT   (void (*)())2    /* force abort on exception */

extern sigfpe_handler_type sigfpe __P((sigfpe_code_type, sigfpe_handler_type));

/*
 * Types for IEEE floating point.
 */
typedef float single;

#ifndef _EXTENDED
#define _EXTENDED
typedef unsigned extended[3];
#endif

typedef long double quadruple;  /* Quadruple-precision type. */

typedef unsigned fp_exception_field_type;
                                /*
                                 * A field containing fp_exceptions OR'ed
                                 * together.
                                 */
/*
 * Definitions for base conversion.
 */
#define DECIMAL_STRING_LENGTH 512       /* Size of buffer in decimal_record. */

typedef char decimal_string[DECIMAL_STRING_LENGTH];
                                /* Decimal significand. */

typedef struct {
        enum fp_class_type fpclass;
        int     sign;
        int     exponent;
        decimal_string ds;      /* Significand - each char contains an ascii */
                                /* digit, except the string-terminating */
                                /* ascii null. */
        int     more;           /* On conversion from decimal to binary, != 0 */
                                /* indicates more non-zero digits following */
                                /* ds. */
        int     ndigits;        /* On fixed_form conversion from binary to */
                                /* decimal, contains number of digits */
                                /* required for ds. */
} decimal_record;

enum decimal_form {
        fixed_form,             /* Fortran F format: ndigits specifies number */
                                /* of digits after point; if negative, */
                                /* specifies rounding to occur to left of */
                                /* point. */
        floating_form           /* Fortran E format: ndigits specifies number */
                                /* of significant digits. */
};
```

```
typedef struct {
        enum fp_direction_type rd;
                                /* Rounding direction. */
        enum decimal_form df;   /* Format for conversion from binary to */
                                /* decimal. */
        int ndigits;            /* Number of digits for conversion. */
} decimal_mode;

enum decimal_string_form {      /* Valid decimal number string formats. */
        invalid_form,           /* Not a valid decimal string format. */
        whitespace_form,        /* All white space - valid in Fortran! */
        fixed_int_form,         /* <digs>               */
        fixed_intdot_form,      /* <digs>.              */
        fixed_dotfrac_form,     /* .<digs>              */
        fixed_intdotfrac_form,  /* <digs>.<frac>        */
        floating_int_form,      /* <digs><exp>          */
        floating_intdot_form,   /* <digs>.<exp>         */
        floating_dotfrac_form,  /* .<digs><exp>         */
        floating_intdotfrac_form, /* <digs>.<digs><exp> */
        inf_form,               /* inf                  */
        infinity_form,          /* infinity             */
        nan_form,               /* nan                  */
        nanstring_form          /* nan(string)          */
};

extern void single_to_decimal __P((single *, decimal_mode *, decimal_record *,
                                fp_exception_field_type *));
extern void double_to_decimal __P((double *, decimal_mode *, decimal_record *,
                                fp_exception_field_type *));
extern void extended_to_decimal __P((extended *, decimal_mode *,
                                decimal_record *, fp_exception_field_type *));
extern void quadruple_to_decimal __P((quadruple *, decimal_mode *,
                                decimal_record *, fp_exception_field_type *));

extern void decimal_to_single __P((single *, decimal_mode *, decimal_record *,
                                fp_exception_field_type *));
extern void decimal_to_double __P((double *, decimal_mode *, decimal_record *,
                                fp_exception_field_type *));
extern void decimal_to_extended __P((extended *, decimal_mode *,
                                decimal_record *, fp_exception_field_type *));
extern void decimal_to_quadruple __P((quadruple *, decimal_mode *,
                                decimal_record *, fp_exception_field_type *));

extern void string_to_decimal __P((char **, int, int, decimal_record *,
                                enum decimal_string_form *, char **));
extern void func_to_decimal __P((char **, int, int, decimal_record *,
                                enum decimal_string_form *, char **,
                                int (*)(void), int *, int (*)(int)));
extern void file_to_decimal __P((char **, int, int, decimal_record *,
                                enum decimal_string_form *, char **,
                                FILE *, int *));

extern char *seconvert __P((single *, int, int *, int *, char *));
extern char *sfconvert __P((single *, int, int *, int *, char *));
extern char *sgconvert __P((single *, int, int, char *));
extern char *econvert __P((double, int, int *, int *, char *));
extern char *fconvert __P((double, int, int *, int *, char *));
extern char *gconvert __P((double, int, int, char *));
extern char *qeconvert __P((quadruple *, int, int *, int *, char *));
extern char *qfconvert __P((quadruple *, int, int *, int *, char *));
extern char *qgconvert __P((quadruple *, int, int, char *));

extern char *ecvt __P((double, int, int *, int *));
extern char *fcvt __P((double, int, int *, int *));
extern char *gcvt __P((double, int, char *));
```



```
#if __cplusplus >= 199711L
namespace std {
#endif
/*
 * ANSI C Standard says the following entry points should be
 * prototyped in <stdlib.h>.  They are now, but weren't before.
 */
extern double atof __P((const char *));
extern double strtod __P((const char *, char **));
#if __cplusplus >= 199711L
}

using std::atof;
using std::strtod;
#endif /* end of namespace std */

#ifdef __cplusplus
}
#endif

#endif /* _FLOATINGPOINT_H */
```

```
**********************************************************
   13589 Thu Oct  9 19:48:51 2014
new/usr/src/lib/Makefile
remove noop change in lib/Makefile
patch01 - 693 import Sun Devpro Math Library
remove noop change in lib/Makefile
patch01 - 693 import Sun Devpro Math Library
**********************************************************
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" 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 (c) 1989, 2010, Oracle and/or its affiliates. All rights reserved.
# Copyright (c) 2012 by Delphix. All rights reserved.
# Copyright (c) 2012, Joyent, Inc. All rights reserved.
# Copyright (c) 2013 Gary Mills

include ../Makefile.master

#       Note that libcurses installs commands along with its library.
#       This is a minor bug which probably should be fixed.
#       Note also that a few extra libraries are kept in cmd source.
#
# Certain libraries are linked with, hence depend on, other libraries.
#
# Although we have historically used .WAIT to express dependencies, it
# reduces the amount of parallelism and thus lengthens the time it
# takes to build the libraries.  Thus, we now require that any new
# libraries explicitly call out their dependencies.  Eventually, all
# the library dependencies will be called out explicitly.  See
# "Library interdependencies" near the end of this file.
#
# Aside from explicit dependencies (and legacy .WAITs), all libraries
# are built in parallel.
#
.PARALLEL:

SUBDIRS= \
        common                  .WAIT   \
        ../cmd/sgs/libconv              \
        ../cmd/sgs/libdl        .WAIT

SUBDIRS += \
        libc                    .WAIT   \
        ../cmd/sgs/libelf       .WAIT   \
        c_synonyms                      \
        libmd                           \
        libmd5                          \
```



```
        librsm                          \
        libmp                   .WAIT   \
        libnsl                  \
        libsecdb                .WAIT   \
        librpcsvc                       \
        libsocket               .WAIT   \
        libsctp                 \
        libsip                  \
        libcommputil            \
        libresolv                       \
        libresolv2              .WAIT   \
        libw                    .WAIT   \
        libintl                 .WAIT   \
        ../cmd/sgs/librtld_db   \
        libaio                  \
        libast                  \
        libdll                  \
        libcmd                  \
        libshell                \
        libsum                  \
        librt                   \
        libadm                  \
        libctf                  \
        libdtrace               \
        libdtrace_jni           \
        libcurses               \
        libtermcap              \
        libgen                  \
        libgss                  \
        libpam                  \
        libuuid                 \
        libthread               \
        libpthread      .WAIT   \
        libslp          \
        libbsdmalloc    \
        libdoor         \
        libdevinfo      \
        libdladm        \
        libdlpi         \
        libeti          \
        libcrypt        \
        libdns_sd       \
        libefi          \
        libfstyp        \
        libwanboot      \
        libwanbootutil  \
        libcryptoutil   \
        libinetutil     \
        libipadm        \
        libipd          \
        libipmp         \
        libiscsit       \
        libkmf          \
        libkstat        \
        libkvm          \
        liblm           \
        libmalloc       \
        libmapmalloc    \
        libmtmalloc     \
        libnls          \
        libnwam         \
        libsmbios       \
        libtecla        \
        libumem         \
        libnvpair       .WAIT   \
        libexacct       \
```

```
125         libsasl         \
126         libldap5        \
127         libsldap        .WAIT   \
128         libbsm          \
129         libsys          \
130         libsysevent     \
131         libnisdb        \
132         libpool         \
133         libpp           \
134         libproc         \
135         libproject      \
136         libsendfile     \
137         nametoaddr      \
138         ncad_addr       \
139         hbaapi          \
140         smhba           \
141         sun_fc          \
142         sun_sas         \
143         gss_mechs/mech_krb5     .WAIT   \
144         libkrb5 .WAIT   \
145         krb5    .WAIT   \
146         libsmbfs        \
147         libfcoe         \
148         libsrpt         \
149         libstmf         \
150         libstmfproxy    \
151         libnsctl        \
152         libunistat      \
153         libdscfg        \
154         librdc          \
155         libinstzones    \
156         libpkg          \
157         libpcidb        \
158         libm1           \
159         libm            \
160         libmvec
157         libpcidb


163 SUBDIRS += \
164         passwdutil      \
165         pam_modules     \
166         crypt_modules   \
167         libadt_jni      \
168         abi             \
169         auditd_plugins  \
170         libvolmgt       \
171         libdevice       \
172         libdevid        \
173         libdhcpsvc      \
174         libc_db         \
175         libndmp         \
176         libsec          \
177         libtnfprobe     \
178         libtnf          \
179         libtnfctl       \
180         libdhcpagent    \
181         libdhcpdu       \
182         libdhcputil     \
183         libxnet         \
184         libipsecutil    \
185         nsswitch        \
186         print           \
187         libuutil        \
188         libscf          \
189         libinetsvc      \
```



```
190         librestart      \
191         libsched        \
192         libelfsign      \
193         pkcs11          .WAIT   \
194         libpctx         .WAIT   \
195         libcpc          \
196         getloginx       \
197         watchmalloc     \
198         extendedFILE    \
199         madv            \
200         mpss            \
201         libdisasm       \
202         libwrap         \
203         libxcurses      \
204         libxcurses2     \
205         libbrand        .WAIT   \
206         libzonecfg      \
207         libzoneinfo     \
208         libzonestat     \
209         libtsnet        \
210         libtsol         \
211         gss_mechs/mech_spnego   \
212         gss_mechs/mech_dummy    \
213         gss_mechs/mech_dh       \
214         rpcsec_gss      \
215         libraidcfg      .WAIT   \
216         librcm          .WAIT   \
217         libcfgadm       .WAIT   \
218         libpicl         .WAIT   \
219         libpicltree     .WAIT \
220         raidcfg_plugins \
221         cfgadm_plugins  \
222         libmail         \
223         lvm             \
224         libsmedia       \
225         libipp          \
226         libdiskmgt      \
227         liblgrp         \
228         libfsmgt        \
229         fm              \
230         libavl          \
231         libcmdutils     \
232         libcontract     \
233         ../cmd/sendmail/libmilter       \
234         sasl_plugins    \
235         udapl           \
236         libzpool        \
237         libzfs_core     \
238         libzfs          \
239         libbe           \
240         pylibbe         \
241         libzfs_jni      \
242         pyzfs           \
243         pysolaris       \
244         libmapid        \
245         brand           \
246         policykit       \
247         hal             \
248         libshare        \
249         libsqlite       \
250         libidmap        \
251         libadutils      \
252         libipmi         \
253         libexacct/demo  \
254         libvrrpadm      \
255         libvscan        \
```

```
256         libgrubmgmt     \
257         smbsrv          \
258         libilb          \
259         scsi            \
260         libima          \
261         libsun_ima      \
262         mpapi           \
263         librstp         \
264         libreparse      \
265         libhotplug      \
266         libfruutils     .WAIT   \
267         libfru          \
268         $($(MACH)_SUBDIRS)

270 i386_SUBDIRS=           \
271         libfdisk        \
272         libsaveargs

274 sparc_SUBDIRS= .WAIT    \
275         efcode          \
276         libds           \
277         libdscp         \
278         libprtdiag      .WAIT   \
279         libprtdiag_psr  \
280         libpri          \
281         librsc          \
282         storage         \
283         libpcp          \
284         libtsalarm      \
285         libv12n

287 FM_sparc_DEPLIBS= libpri

289 fm:                     \
290         libexacct       \
291         libipmi         \
292         libzfs          \
293         scsi            \
294         $(FM_$(MACH)_DEPLIBS)

296 #
297 # Create a special version of $(SUBDIRS) with no .WAIT's, for use with the
298 # clean and clobber targets (for more information, see those targets, below).
299 #
300 NOWAIT_SUBDIRS= $(SUBDIRS:.WAIT=)

302 DCSUBDIRS =             \
303         lvm

305 MSGSUBDIRS=             \
306         abi             \
307         auditd_plugins  \
308         brand           \
309         cfgadm_plugins  \
310         gss_mechs/mech_dh       \
311         gss_mechs/mech_krb5     \
312         krb5            \
313         libast          \
314         libbsm          \
315         libc            \
316         libcfgadm       \
317         libcmd          \
318         libcontract     \
319         libcurses       \
320         libdhcpsvc      \
321         libdhcputil     \
```



```
322         libipsecutil    \
323         libdiskmgt      \
324         libdladm        \
325         libdll          \
326         libgrubmgmt     \
327         libgss          \
328         libidmap        \
329         libipmp         \
330         libilb          \
331         libinetutil     \
332         libinstzones    \
333         libipadm        \
334         libnsl          \
335         libnwam         \
336         libpam          \
337         libpicl         \
338         libpool         \
339         libpkg          \
340         libpp           \
341         libscf          \
342         libsasl         \
343         libldap5        \
344         libsecdb        \
345         libshare        \
346         libshell        \
347         libsldap        \
348         libslp          \
349         libsmbfs        \
350         libsmedia       \
351         libsum          \
352         libtsol         \
353         libuutil        \
354         libvrrpadm      \
355         libvscan        \
356         libwanboot      \
357         libwanbootutil  \
358         libzfs          \
359         libzonecfg      \
360         lvm             \
361         madv            \
362         mpss            \
363         pam_modules     \
364         pyzfs           \
365         pysolaris       \
366         rpcsec_gss      \
367         libreparse
368 MSGSUBDIRS += \
369         $($(MACH)_MSGSUBDIRS)

371 sparc_MSGSUBDIRS=       \
372         libprtdiag      \
373         libprtdiag_psr

375 i386_MSGSUBDIRS= libfdisk

377 HDRSUBDIRS=             \
378         auditd_plugins  \
379         libast          \
380         libbrand        \
381         libbsm          \
382         libc            \
383         libcmd          \
384         libcmdutils     \
385         libcommputil    \
386         libcontract     \
387         libcpc          \
```

```
388         libctf          \
389         libcurses       \
390         libtermcap      \
391         libcryptoutil   \
392         libdevice       \
393         libdevid        \
394         libdevinfo      \
395         libdiskmgt      \
396         libdladm        \
397         libdll          \
398         libdlpi         \
399         libdhcpagent    \
400         libdhcpsvc      \
401         libdhcputil     \
402         libdisasm       \
403         libdns_sd       \
404         libdscfg        \
405         libdtrace       \
406         libdtrace_jni   \
407         libelfsign      \
408         libeti          \
409         libfru          \
410         libfstyp        \
411         libgen          \
412         libipadm        \
413         libipd          \
414         libipsecutil    \
415         libinetsvc      \
416         libinetutil     \
417         libinstzones    \
418         libipmi         \
419         libipmp         \
420         libipp          \
421         libiscsit       \
422         libkstat        \
423         libkvm          \
424         libmail         \
425         libmd           \
426         libmtmalloc     \
427         libndmp         \
428         libnvpair       \
429         libnsctl        \
430         libnsl          \
431         libnwam         \
432         libpam          \
433         libpcidb        \
434         libpctx         \
435         libpicl         \
436         libpicltree     \
437         libpool         \
438         libpp           \
439         libproc         \
440         libraidcfg      \
441         librcm          \
442         librdc          \
443         libscf          \
444         libsip          \
445         libsmbios       \
446         librestart      \
447         librpcsvc       \
448         librsm          \
449         librstp         \
450         libsasl         \
451         libsec          \
452         libshell        \
453         libslp          \
```



```
454         libsmedia       \
455         libsocket       \
456         libsqlite       \
457         libfcoe         \
458         libsrpt         \
459         libstmf         \
460         libstmfproxy    \
461         libsum          \
462         libsysevent     \
463         libtecla        \
464         libtnf          \
465         libtnfctl       \
466         libtnfprobe     \
467         libtsnet        \
468         libtsol         \
469         libvrrpadm      \
470         libvolmgt       \
471         libumem         \
472         libunistat      \
473         libuutil        \
474         libwanboot      \
475         libwanbootutil  \
476         libwrap         \
477         libxcurses2     \
478         libzfs          \
479         libzfs_core     \
480         libzfs_jni      \
481         libzoneinfo     \
482         libzonestat     \
483         hal             \
484         policykit       \
485         lvm             \
486         pkcs11          \
487         passwdutil      \
488         ../cmd/sendmail/libmilter       \
489         fm              \
490         udapl           \
491         libmapid        \
492         libkrb5         \
493         libsmbfs        \
494         libshare        \
495         libidmap        \
496         libvscan        \
497         libgrubmgmt     \
498         smbsrv          \
499         libilb          \
500         scsi            \
501         hbaapi          \
502         smhba           \
503         libima          \
504         libsun_ima      \
505         mpapi           \
506         libreparse      \
507         $($(MACH)_HDRSUBDIRS)

509 i386_HDRSUBDIRS=       \
510         libfdisk        \
511         libsaveargs

513 sparc_HDRSUBDIRS=       \
514         libds           \
515         libdscp         \
516         libpri          \
517         libv12n         \
518         storage
```

```
520 all :=          TARGET= all
521 check :=        TARGET= check
522 clean :=        TARGET= clean
523 clobber :=      TARGET= clobber
524 install :=      TARGET= install
525 install_h :=    TARGET= install_h
526 lint :=         TARGET= lint
527 _dc :=          TARGET= _dc
528 _msg :=         TARGET= _msg

530 .KEEP_STATE:

532 #
533 # For the all and install targets, we clearly must respect library
534 # dependencies so that the libraries link correctly.  However, for
535 # the remaining targets (check, clean, clobber, install_h, lint, _dc
536 # and _msg), libraries do not have any dependencies on one another
537 # and thus respecting dependencies just slows down the build.
538 # As such, for these rules, we use pattern replacement to explicitly
539 # avoid triggering the dependency information.  Note that for clean,
540 # clobber and lint, we must use $(NOWAIT_SUBDIRS) rather than
541 # $(SUBDIRS), to prevent '.WAIT' from expanding to '.WAIT-nodepend'.
542 #

544 all:                    $(SUBDIRS)

546 install:                $(SUBDIRS) .WAIT install_extra

548 # extra libraries kept in other source areas
549 install_extra:
550         @cd ../cmd/sgs; pwd; $(MAKE) install_lib
551         @pwd

553 clean clobber lint:     $(NOWAIT_SUBDIRS:%=%-nodepend)

555 install_h check:        $(HDRSUBDIRS:%=%-nodepend)

557 _msg:                   $(MSGSUBDIRS:%=%-nodepend) .WAIT _dc

559 _dc:                    $(DCSUBDIRS:%=%-nodepend)

561 #
562 # Library interdependencies are called out explicitly here
563 #
564 auditd_plugins: libbsm libnsl libsecdb
565 gss_mechs/mech_krb5:    libgss libnsl libsocket libresolv pkcs11
566 libadt_jni:     libbsm
567 libast:         libsocket libm
563 libast:         libsocket
568 libadutils:     libldap5 libresolv libsocket libnsl
569 nsswitch:       libadutils libidmap
570 libbe:          libzfs
571 libbsm:         libtsol
572 libcmd:         libsum libast libsocket libnsl
573 libcmdutils:    libavl
574 libcontract:    libnvpair
575 libdevid:       libdevinfo
576 libdevinfo:     libnvpair libsec
577 libdhcpagent:   libsocket libdhcputil libuuid libdlpi libcontract
578 libdhcpsvc:     libinetutil
579 libdhcputil:    libnsl libgen libinetutil libdlpi
580 libdladm:       libdevinfo libinetutil libsocket libscf librcm libnvpair \
581                 libexacct libnsl libkstat libcurses
582 libdll:         libast
583 libdlpi:        libinetutil libdladm
584 libds:          libsysevent
```



```
585 libdscfg:       libnsctl libunistat libsocket libnsl
586 libdtrace:      libproc libgen libctf
587 libdtrace_jni:  libuutil libdtrace
588 libefi:         libuuid
589 libfstyp:       libnvpair
590 libelfsign:     libcryptoutil libkmf
591 libidmap:       libadutils libldap5 libavl libsldap libuutil
592 libipadm:       libnsl libinetutil libsocket libdlpi libnvpair libdhcpagent \
593                 libdladm libsecdb
594 libiscsit:      libc libnvpair libstmf libuuid libnsl
595 libkmf:         libcryptoutil pkcs11
596 libm:           libc
597 libm1:          libc libm
598 libmvec:        libc libm
599 libnsl:         libmd5
600 libmapid:       libresolv
601 librdc:         libsocket libnsl libnsctl libunistat libdscfg
602 libuuid:        libdlpi
603 libinetutil:    libsocket
604 libipsecutil:   libtecla libsocket
605 libinstzones:   libzonecfg libcontract
606 libpkg:         libwanboot libscf libadm
607 libnwam:        libscf
608 libsecdb:       libnsl
609 libsasl:        libgss libsocket pkcs11 libmd
610 sasl_plugins:   pkcs11 libgss libsocket libsasl
611 libsctp:        libsocket
612 libshell:       libast libcmd libdll libsocket libsecdb libm
605 libshell:       libast libcmd libdll libsocket libsecdb
613 libsip:         libmd5
614 libsmbfs:       libcmdutils libsocket libnsl libkrb5
615 libsocket:      libnsl
616 libstmfproxy:   libstmf libsocket libnsl libpthread
617 libsum:         libast
618 libsysevent:    libsecdb
619 libldap5:       libsasl libsocket libnsl libmd
620 libsldap:       libldap5 libtsol libnsl libc libscf libresolv
621 libpool:        libnvpair libexacct
622 libpp:          libast
623 libzonecfg:     libc libsocket libnsl libuuid libnvpair libsysevent libsec \
624                 libbrand libpool libscf
625 libproc:        ../cmd/sgs/librtld_db ../cmd/sgs/libelf libctf libsaveargs
626 libproject:     libpool libproc libsecdb
627 libtermcap:     libcurses
628 libtsnet:       libnsl libtsol libsecdb
629 libwrap:        libnsl libsocket
630 libwanboot:     libnvpair libresolv libnsl libsocket libdevinfo libinetutil \
631                 libdhcputil
632 libwanbootutil: libnsl
633 pam_modules:    libproject passwdutil smbsrv
634 libscf:         libuutil libmd libgen libsmbios libnsl
635 libinetsvc:     libscf
636 librestart:     libuutil libscf
637 libsaveargs:    libdisasm
638 ../cmd/sgs/libdl:       ../cmd/sgs/libconv
639 ../cmd/sgs/libelf:      ../cmd/sgs/libconv
640 pkcs11:         libcryptoutil
641 print:          libldap5
642 udapl/udapl_tavor:      udapl/libdat
643 libzfs:         libdevid libgen libnvpair libuutil \
644                 libadm libavl libefi libidmap libmd libzfs_core libm
637                 libadm libavl libefi libidmap libmd libzfs_core
645 libzfs_core:    libnvpair
646 libzfs_jni:     libdiskmgt libnvpair libzfs
647 libzpool:       libavl libumem libnvpair libcmdutils
648 libsec:         libavl libidmap
```

```
649 brand:          libc libsocket
650 libshare:       libscf libzfs libuuid libfsmgt libsecdb libumem libsmbfs
651 libexacct/demo: libexacct libproject libsocket libnsl
652 libtsalarm:     libpcp
653 smbsrv:         libsocket libnsl libmd libxnet libpthread librt \
654                 libshare libidmap pkcs11 libsqlite libcryptoutil \
655                 libreparse libcmdutils
656 libv12n:        libds libuuid
657 libvrrpadm:     libsocket libdladm libscf
658 libvscan:       libscf
659 libfru:         libfruutils
660 scsi:           libnvpair libfru
661 mpapi:          libpthread libdevinfo libsysevent libnvpair
662 sun_fc:         libdevinfo libsysevent libnvpair
663 libsun_ima:     libdevinfo libsysevent libnsl
664 sun_sas:        libdevinfo libsysevent libnvpair libkstat libdevid
665 libgrubmgmt:    libdevinfo libzfs libfstyp
666 pylibbe:        libbe libzfs
667 pyzfs:          libnvpair libzfs
668 pysolaris:      libsec libidmap
669 libreparse:     libnvpair
670 libhotplug:     libnvpair
671 cfgadm_plugins: libhotplug
672 libilb:         libsocket
673 libipmi:        libm
674 libprtdiag:     libm
675 libsqlite:      libm
676 libstmf:        libm
677 libvscan:       libm


680 $(INTEL_BUILD)libdiskmgt:libfdisk

682 #
683 # The reason this rule checks for the existence of the
684 # Makefile is that some of the directories do not exist
685 # in certain situations (e.g., exportable source builds,
686 # OpenSolaris).
687 #
688 $(SUBDIRS): FRC
689         @if [ -f $@/Makefile  ]; then \
690                 cd $@; pwd; $(MAKE) $(TARGET); \
691         else \
692                 true; \
693         fi

695 $(SUBDIRS:%=%-nodepend):
696         @if [ -f $(@:%-nodepend=%)/Makefile  ]; then \
697                 cd $(@:%-nodepend=%); pwd; $(MAKE) $(TARGET); \
698         else \
699                 true; \
700         fi

702 FRC:
```

```
**********************************************************
   19628 Thu Oct  9 19:48:52 2014
new/usr/src/lib/libm/Makefile.com
remove -Wno-switch -Wno-parentheses -Wno-unused-variable from libm
remove -Wno-uninitialized for libm
libm fixes from richlowe - richlowe.net/webrevs/il_keith
patch01 - 693 import Sun Devpro Math Library
remove -Wno-switch -Wno-parentheses -Wno-unused-variable from libm
#4625 add cscope.out to the .gitignore
remove -Wno-uninitialized for libm
libm fixes from richlowe - richlowe.net/webrevs/il_keith
patch01 - 693 import Sun Devpro Math Library
**********************************************************
#
# 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 Nexenta Systems, Inc.  All rights reserved.
#

LIBRARY         = libm.a
VERS            = .2

LIBMDIR         = $(SRC)/lib/libm

m9xsseOBJS_i386 = \
                  __fex_hdlr.o \
                  __fex_i386.o \
                  __fex_sse.o \
                  __fex_sym.o \
                  fex_log.o

m9xsseOBJS      = $(m9xsseOBJS_$(TARGET_ARCH))

m9xOBJS_amd64   = \
                  __fex_sse.o \
                  feprec.o

m9xOBJS_sparc   = \
                  lrint.o \
                  lrintf.o \
                  lrintl.o \
                  lround.o \
                  lroundf.o \
                  lroundl.o

m9xOBJS_i386    = \
                  __fex_sse.o \
                  feprec.o \
                  lrint.o \
                  lrintf.o \
                  lrintl.o \
                  lround.o \
                  lroundf.o \
                  lroundl.o

#
# lrint.o, lrintf.o, lrintl.o, lround.o, lroundf.o & lroundl.o are 32-bit only
```



```
#
m9xOBJS         = \
                  $(m9xOBJS_$(TARGET_ARCH)) \
                  __fex_$(MACH).o \
                  __fex_hdlr.o \
                  __fex_sym.o \
                  fdim.o \
                  fdimf.o \
                  fdiml.o \
                  feexcept.o \
                  fenv.o \
                  feround.o \
                  fex_handler.o \
                  fex_log.o \
                  fma.o \
                  fmaf.o \
                  fmal.o \
                  fmax.o \
                  fmaxf.o \
                  fmaxl.o \
                  fmin.o \
                  fminf.o \
                  fminl.o \
                  frexp.o \
                  frexpf.o \
                  frexpl.o \
                  ldexp.o \
                  ldexpf.o \
                  ldexpl.o \
                  llrint.o \
                  llrintf.o \
                  llrintl.o \
                  llround.o \
                  llroundf.o \
                  llroundl.o \
                  modf.o \
                  modff.o \
                  modfl.o \
                  nan.o \
                  nanf.o \
                  nanl.o \
                  nearbyint.o \
                  nearbyintf.o \
                  nearbyintl.o \
                  nexttoward.o \
                  nexttowardf.o \
                  nexttowardl.o \
                  remquo.o \
                  remquof.o \
                  remquol.o \
                  round.o \
                  roundf.o \
                  roundl.o \
                  scalbln.o \
                  scalblnf.o \
                  scalblnl.o \
                  tgamma.o \
                  tgammaf.o \
                  tgammal.o \
                  trunc.o \
                  truncf.o \
                  truncl.o

OBJS_M9XSSE     = $(m9xsseOBJS:%=pics/%)

COBJS_i386      = \
```

```
120                 __libx_errno.o

122 COBJS_sparc     = \
123                 $(COBJS_i386) \
124                 _TBL_atan.o \
125                 _TBL_exp2.o \
126                 _TBL_log.o \
127                 _TBL_log2.o \
128                 _TBL_tan.o \
129                 __tan.o \
130                 __tanf.o

132 #
133 # atan2pi.o and sincospi.o is for internal use only
134 #

136 COBJS_amd64 = \
137                 _TBL_atan.o \
138                 _TBL_exp2.o \
139                 _TBL_log.o \
140                 _TBL_log2.o \
141                 __tan.o \
142                 __tanf.o \
143                 _TBL_tan.o \
144                 copysign.o \
145                 exp.o \
146                 fabs.o \
147                 fmod.o \
148                 ilogb.o \
149                 isnan.o \
150                 nextafter.o \
151                 remainder.o \
152                 rint.o \
153                 scalbn.o

155 COBJS_sparcv9 = $(COBJS_amd64)

157 COBJS           = \
158                 $(COBJS_$(TARGET_ARCH)) \
159                 __cos.o \
160                 __lgamma.o \
161                 __rem_pio2.o \
162                 __rem_pio2m.o \
163                 __sin.o \
164                 __sincos.o \
165                 __xpg6.o \
166                 _lib_version.o \
167                 _SVID_error.o \
168                 _TBL_ipio2.o \
169                 _TBL_sin.o \
170                 acos.o \
171                 acosh.o \
172                 asin.o \
173                 asinh.o \
174                 atan.o \
175                 atan2.o \
176                 atan2pi.o \
177                 atanh.o \
178                 cbrt.o \
179                 ceil.o \
180                 cos.o \
181                 cosh.o \
182                 erf.o \
183                 exp10.o \
184                 exp2.o \
185                 expm1.o \
```



```
186                 floor.o \
187                 gamma.o \
188                 gamma_r.o \
189                 hypot.o \
190                 j0.o \
191                 j1.o \
192                 jn.o \
193                 lgamma.o \
194                 lgamma_r.o \
195                 log.o \
196                 log10.o \
197                 log1p.o \
198                 log2.o \
199                 logb.o \
200                 matherr.o \
201                 pow.o \
202                 scalb.o \
203                 signgam.o \
204                 significand.o \
205                 sin.o \
206                 sincos.o \
207                 sincospi.o \
208                 sinh.o \
209                 sqrt.o \
210                 tan.o \
211                 tanh.o

213 #
214 # LSARC/2003/658 adds isnanl
215 #
216 QOBJS_sparc     = \
217                 _TBL_atanl.o \
218                 _TBL_expl.o \
219                 _TBL_expm1l.o \
220                 _TBL_logl.o \
221                 finitel.o \
222                 isnanl.o

224 QOBJS_sparcv9   = $(QOBJS_sparc)

226 QOBJS_amd64     = \
227                 finitel.o \
228                 isnanl.o

230 #
231 # atan2pil.o, ieee_funcl.o, rndintl.o, sinpil.o, sincospil.o
232 # are for internal use only
233 #
234 # LSARC/2003/279 adds the following:
235 #               gammal.o        1
236 #               gammal_r.o      1
237 #               j0l.o           2
238 #               j1l.o           2
239 #               jnl.o           2
240 #               lgammal_r.o     1
241 #               scalbl.o        1
242 #               significandl.o  1
243 #
244 QOBJS           = \
245                 $(QOBJS_$(TARGET_ARCH)) \
246                 __cosl.o \
247                 __lgammal.o \
248                 __poly_libmq.o \
249                 __rem_pio2l.o \
250                 __sincosl.o \
251                 __sinl.o \
```

```
252                 __tanl.o \
253                 _TBL_cosl.o \
254                 _TBL_ipio2l.o \
255                 _TBL_sinl.o \
256                 _TBL_tanl.o \
257                 acoshl.o \
258                 acosl.o \
259                 asinhl.o \
260                 asinl.o \
261                 atan2l.o \
262                 atan2pil.o \
263                 atanhl.o \
264                 atanl.o \
265                 cbrtl.o \
266                 copysignl.o \
267                 coshl.o \
268                 cosl.o \
269                 erfl.o \
270                 exp10l.o \
271                 exp2l.o \
272                 expl.o \
273                 expm1l.o \
274                 fabsl.o \
275                 floorl.o \
276                 fmodl.o \
277                 gammal.o \
278                 gammal_r.o \
279                 hypotl.o \
280                 ieee_funcl.o \
281                 ilogbl.o \
282                 j0l.o \
283                 j1l.o \
284                 jnl.o \
285                 lgammal.o \
286                 lgammal_r.o \
287                 log10l.o \
288                 log1pl.o \
289                 log2l.o \
290                 logbl.o \
291                 logl.o \
292                 nextafterl.o \
293                 powl.o \
294                 remainderl.o \
295                 rintl.o \
296                 rndintl.o \
297                 scalbl.o \
298                 scalbnl.o \
299                 signgaml.o \
300                 significandl.o \
301                 sincosl.o \
302                 sincospil.o \
303                 sinhl.o \
304                 sinl.o \
305                 sinpil.o \
306                 sqrtl.o \
307                 tanhl.o \
308                 tanl.o

310 #
311 # LSARC/2003/658 adds isnanf
312 #
313 ROBJS_sparc     = \
314                 __cosf.o \
315                 __sincosf.o \
316                 __sinf.o \
317                 isnanf.o
```



```
319 ROBJS_sparcv9   = $(ROBJS_sparc)

321 ROBJS_amd64     = \
322                 isnanf.o \
323                 __cosf.o \
324                 __sincosf.o \
325                 __sinf.o

327 #
328 # atan2pif.o, sincosf.o, sincospif.o are for internal use only
329 #
330 # LSARC/2003/279 adds the following:
331 #               besself.o       6
332 #               scalbf.o        1
333 #               gammaf.o        1
334 #               gammaf_r.o      1
335 #               lgammaf_r.o     1
336 #               significandf.o  1
337 #
338 ROBJS           = \
339                 $(ROBJS_$(TARGET_ARCH)) \
340                 _TBL_r_atan_.o \
341                 acosf.o \
342                 acoshf.o \
343                 asinf.o \
344                 asinhf.o \
345                 atan2f.o \
346                 atan2pif.o \
347                 atanf.o \
348                 atanhf.o \
349                 besself.o \
350                 cbrtf.o \
351                 copysignf.o \
352                 cosf.o \
353                 coshf.o \
354                 erff.o \
355                 exp10f.o \
356                 exp2f.o \
357                 expf.o \
358                 expm1f.o \
359                 fabsf.o \
360                 floorf.o \
361                 fmodf.o \
362                 gammaf.o \
363                 gammaf_r.o \
364                 hypotf.o \
365                 ilogbf.o \
366                 lgammaf.o \
367                 lgammaf_r.o \
368                 log10f.o \
369                 log1pf.o \
370                 log2f.o \
371                 logbf.o \
372                 logf.o \
373                 nextafterf.o \
374                 powf.o \
375                 remainderf.o \
376                 rintf.o \
377                 scalbf.o \
378                 scalbnf.o \
379                 signgamf.o \
380                 significandf.o \
381                 sinf.o \
382                 sinhf.o \
383                 sincosf.o \
```

```
384                 sincospif.o \
385                 sqrtf.o \
386                 tanf.o \
387                 tanhf.o

389 #
390 # LSARC/2003/658 adds isnanf/isnanl
391 #

393 SOBJS_sparc     = \
394                 copysign.o \
395                 exp.o \
396                 fabs.o \
397                 fmod.o \
398                 ilogb.o \
399                 isnan.o \
400                 nextafter.o \
401                 remainder.o \
402                 rint.o \
403                 scalbn.o

405 SOBJS_i386      = \
406                 __reduction.o \
407                 finitef.o \
408                 finitel.o \
409                 isnanf.o \
410                 isnanl.o \
411                 $(SOBJS_sparc)

413 SOBJS_amd64     = \
414                 __swapFLAGS.o
415 #               _xtoll.o \
416 #               _xtoull.o \


419 SOBJS           = \
420                 $(SOBJS_$(TARGET_ARCH))

422 complexOBJS     = \
423                 cabs.o \
424                 cabsf.o \
425                 cabsl.o \
426                 cacos.o \
427                 cacosf.o \
428                 cacosh.o \
429                 cacoshf.o \
430                 cacoshl.o \
431                 cacosl.o \
432                 carg.o \
433                 cargf.o \
434                 cargl.o \
435                 casin.o \
436                 casinf.o \
437                 casinh.o \
438                 casinhf.o \
439                 casinhl.o \
440                 casinl.o \
441                 catan.o \
442                 catanf.o \
443                 catanh.o \
444                 catanhf.o \
445                 catanhl.o \
446                 catanl.o \
447                 ccos.o \
448                 ccosf.o \
449                 ccosh.o \
```



```
450                 ccoshf.o \
451                 ccoshl.o \
452                 ccosl.o \
453                 cexp.o \
454                 cexpf.o \
455                 cexpl.o \
456                 cimag.o \
457                 cimagf.o \
458                 cimagl.o \
459                 clog.o \
460                 clogf.o \
461                 clogl.o \
462                 conj.o \
463                 conjf.o \
464                 conjl.o \
465                 cpow.o \
466                 cpowf.o \
467                 cpowl.o \
468                 cproj.o \
469                 cprojf.o \
470                 cprojl.o \
471                 creal.o \
472                 crealf.o \
473                 creall.o \
474                 csin.o \
475                 csinf.o \
476                 csinh.o \
477                 csinhf.o \
478                 csinhl.o \
479                 csinl.o \
480                 csqrt.o \
481                 csqrtf.o \
482                 csqrtl.o \
483                 ctan.o \
484                 ctanf.o \
485                 ctanh.o \
486                 ctanhf.o \
487                 ctanhl.o \
488                 ctanl.o \
489                 k_atan2.o \
490                 k_atan2l.o \
491                 k_cexp.o \
492                 k_cexpl.o \
493                 k_clog_r.o \
494                 k_clog_rl.o

496 OBJECTS         = $(COBJS) $(ROBJS) $(QOBJS) $(SOBJS) $(m9xOBJS) $(complexOBJS)

498 include         $(SRC)/lib/Makefile.lib
499 include         $(LIBMDIR)/Makefile.libm.com
500 include         $(SRC)/lib/Makefile.rootfs

502 SRCDIR          = ../common/
503 LIBS            = $(DYNLIB) $(LINTLIB)

505 LINTERROFF      = -erroff=E_FUNC_SET_NOT_USED
506 LINTERROFF      += -erroff=E_FUNC_RET_ALWAYS_IGNOR2
507 LINTERROFF      += -erroff=E_FUNC_RET_MAYBE_IGNORED2
508 LINTERROFF      += -erroff=E_IMPL_CONV_RETURN
509 LINTERROFF      += -erroff=E_NAME_MULTIPLY_DEF2
510 LINTFLAGS       += $(LINTERROFF)
511 LINTFLAGS64     += $(LINTERROFF)
512 LINTFLAGS64     += -errchk=longptr64

514 CPPFLAGS        += -DLIBM_BUILD
```

```
CFLAGS          += $(C_BIGPICFLAGS)
CFLAGS64        += $(C_BIGPICFLAGS)

m9x_IL          = $(LIBMDIR)/common/m9x/__fenv_$(TARGET_ARCH).il

SRCS_LD_i386_amd64 = \
        ../common/LD/finitel.c \
        ../common/LD/isnanl.c \
        ../common/LD/nextafterl.c

SRCS_LD = \
        $(SRCS_LD_i386_$(TARGET_ARCH)) \
        ../common/LD/__cosl.c \
        ../common/LD/__lgammal.c \
        ../common/LD/__poly_libmq.c \
        ../common/LD/__rem_pio2l.c \
        ../common/LD/__sincosl.c \
        ../common/LD/__sinl.c \
        ../common/LD/__tanl.c \
        ../common/LD/_TBL_cosl.c \
        ../common/LD/_TBL_ipio2l.c \
        ../common/LD/_TBL_sinl.c \
        ../common/LD/_TBL_tanl.c \
        ../common/LD/acoshl.c \
        ../common/LD/asinhl.c \
        ../common/LD/atan2pil.c \
        ../common/LD/atanhl.c \
        ../common/LD/cbrtl.c \
        ../common/LD/coshl.c \
        ../common/LD/cosl.c \
        ../common/LD/erfl.c \
        ../common/LD/gammal.c \
        ../common/LD/gammal_r.c \
        ../common/LD/hypotl.c \
        ../common/LD/j0l.c \
        ../common/LD/j1l.c \
        ../common/LD/jnl.c \
        ../common/LD/lgammal.c \
        ../common/LD/lgammal_r.c \
        ../common/LD/log1pl.c \
        ../common/LD/logbl.c \
        ../common/LD/scalbl.c \
        ../common/LD/signgaml.c \
        ../common/LD/significandl.c \
        ../common/LD/sincosl.c \
        ../common/LD/sincospil.c \
        ../common/LD/sinhl.c \
        ../common/LD/sinl.c \
        ../common/LD/sinpil.c \
        ../common/LD/tanhl.c \
        ../common/LD/tanl.c

SRCS_LD_i386 = \
        $(SRCS_LD)

SRCS_R_amd64 = \
        ../common/R/__tanf.c \
        ../common/R/isnanf.c \
        ../common/R/__cosf.c \
        ../common/R/__sincosf.c \
        ../common/R/__sinf.c \
        ../common/R/acosf.c \
        ../common/R/asinf.c \
        ../common/R/atan2f.c \
        ../common/R/copysignf.c \
        ../common/R/exp10f.c \
```



```
        ../common/R/exp2f.c \
        ../common/R/expm1f.c \
        ../common/R/fabsf.c \
        ../common/R/hypotf.c \
        ../common/R/ilogbf.c \
        ../common/R/log10f.c \
        ../common/R/log2f.c \
        ../common/R/nextafterf.c \
        ../common/R/powf.c \
        ../common/R/rintf.c \
        ../common/R/scalbnf.c

# sparc + sparcv9
SRCS_R_sparc = \
        ../common/R/__tanf.c \
        ../common/R/__cosf.c \
        ../common/R/__sincosf.c \
        ../common/R/__sinf.c \
        ../common/R/isnanf.c \
        ../common/R/acosf.c \
        ../common/R/asinf.c \
        ../common/R/atan2f.c \
        ../common/R/copysignf.c \
        ../common/R/exp10f.c \
        ../common/R/exp2f.c \
        ../common/R/expm1f.c \
        ../common/R/fabsf.c \
        ../common/R/fmodf.c \
        ../common/R/hypotf.c \
        ../common/R/ilogbf.c \
        ../common/R/log10f.c \
        ../common/R/log2f.c \
        ../common/R/nextafterf.c \
        ../common/R/powf.c \
        ../common/R/remainderf.c \
        ../common/R/rintf.c \
        ../common/R/scalbnf.c

SRCS_R = \
        $(SRCS_R_$(MACH)) \
        $(SRCS_R_$(TARGET_ARCH)) \
        ../common/R/_TBL_r_atan_.c \
        ../common/R/acoshf.c \
        ../common/R/asinhf.c \
        ../common/R/atan2pif.c \
        ../common/R/atanf.c \
        ../common/R/atanhf.c \
        ../common/R/besself.c \
        ../common/R/cbrtf.c \
        ../common/R/cosf.c \
        ../common/R/coshf.c \
        ../common/R/erff.c \
        ../common/R/expf.c \
        ../common/R/floorf.c \
        ../common/R/gammaf.c \
        ../common/R/gammaf_r.c \
        ../common/R/lgammaf.c \
        ../common/R/lgammaf_r.c \
        ../common/R/log1pf.c \
        ../common/R/logbf.c \
        ../common/R/logf.c \
        ../common/R/scalbf.c \
        ../common/R/signgamf.c \
        ../common/R/significandf.c \
        ../common/R/sinf.c \
        ../common/R/sinhf.c \
```

```
648         ../common/R/sincosf.c \
649         ../common/R/sincospif.c \
650         ../common/R/sqrtf.c \
651         ../common/R/tanf.c \
652         ../common/R/tanhf.c

654 SRCS_Q = \
655         ../common/Q/_TBL_atanl.c \
656         ../common/Q/_TBL_expl.c \
657         ../common/Q/_TBL_expm1l.c \
658         ../common/Q/_TBL_logl.c \
659         ../common/Q/finitel.c \
660         ../common/Q/isnanl.c \
661         ../common/Q/__cosl.c \
662         ../common/Q/__lgammal.c \
663         ../common/Q/__poly_libmq.c \
664         ../common/Q/__rem_pio2l.c \
665         ../common/Q/__sincosl.c \
666         ../common/Q/__sinl.c \
667         ../common/Q/__tanl.c \
668         ../common/Q/_TBL_cosl.c \
669         ../common/Q/_TBL_ipio2l.c \
670         ../common/Q/_TBL_sinl.c \
671         ../common/Q/_TBL_tanl.c \
672         ../common/Q/acoshl.c \
673         ../common/Q/acosl.c \
674         ../common/Q/asinhl.c \
675         ../common/Q/asinl.c \
676         ../common/Q/atan2l.c \
677         ../common/Q/atan2pil.c \
678         ../common/Q/atanhl.c \
679         ../common/Q/atanl.c \
680         ../common/Q/cbrtl.c \
681         ../common/Q/copysignl.c \
682         ../common/Q/coshl.c \
683         ../common/Q/cosl.c \
684         ../common/Q/erfl.c \
685         ../common/Q/exp10l.c \
686         ../common/Q/exp2l.c \
687         ../common/Q/expl.c \
688         ../common/Q/expm1l.c \
689         ../common/Q/fabsl.c \
690         ../common/Q/floorl.c \
691         ../common/Q/fmodl.c \
692         ../common/Q/gammal.c \
693         ../common/Q/gammal_r.c \
694         ../common/Q/hypotl.c \
695         ../common/Q/ieee_funcl.c \
696         ../common/Q/ilogbl.c \
697         ../common/Q/j0l.c \
698         ../common/Q/j1l.c \
699         ../common/Q/jnl.c \
700         ../common/Q/lgammal.c \
701         ../common/Q/lgammal_r.c \
702         ../common/Q/log10l.c \
703         ../common/Q/log1pl.c \
704         ../common/Q/log2l.c \
705         ../common/Q/logbl.c \
706         ../common/Q/logl.c \
707         ../common/Q/nextafterl.c \
708         ../common/Q/powl.c \
709         ../common/Q/remainderl.c \
710         ../common/Q/rintl.c \
711         ../common/Q/rndintl.c \
712         ../common/Q/scalbl.c \
713         ../common/Q/scalbnl.c \
```



```
714         ../common/Q/signgaml.c \
715         ../common/Q/significandl.c \
716         ../common/Q/sincosl.c \
717         ../common/Q/sincospil.c \
718         ../common/Q/sinhl.c \
719         ../common/Q/sinl.c \
720         ../common/Q/sinpil.c \
721         ../common/Q/sqrtl.c \
722         ../common/Q/tanhl.c \
723         ../common/Q/tanl.c

725 SRCS_Q_sparc = \
726         $(SRCS_Q)

728 SRCS_complex = \
729         ../common/complex/cabs.c \
730         ../common/complex/cabsf.c \
731         ../common/complex/cabsl.c \
732         ../common/complex/cacos.c \
733         ../common/complex/cacosf.c \
734         ../common/complex/cacosh.c \
735         ../common/complex/cacoshf.c \
736         ../common/complex/cacoshl.c \
737         ../common/complex/cacosl.c \
738         ../common/complex/carg.c \
739         ../common/complex/cargf.c \
740         ../common/complex/cargl.c \
741         ../common/complex/casin.c \
742         ../common/complex/casinf.c \
743         ../common/complex/casinh.c \
744         ../common/complex/casinhf.c \
745         ../common/complex/casinhl.c \
746         ../common/complex/casinl.c \
747         ../common/complex/catan.c \
748         ../common/complex/catanf.c \
749         ../common/complex/catanh.c \
750         ../common/complex/catanhf.c \
751         ../common/complex/catanhl.c \
752         ../common/complex/catanl.c \
753         ../common/complex/ccos.c \
754         ../common/complex/ccosf.c \
755         ../common/complex/ccosh.c \
756         ../common/complex/ccoshf.c \
757         ../common/complex/ccoshl.c \
758         ../common/complex/ccosl.c \
759         ../common/complex/cexp.c \
760         ../common/complex/cexpf.c \
761         ../common/complex/cexpl.c \
762         ../common/complex/cimag.c \
763         ../common/complex/cimagf.c \
764         ../common/complex/cimagl.c \
765         ../common/complex/clog.c \
766         ../common/complex/clogf.c \
767         ../common/complex/clogl.c \
768         ../common/complex/conj.c \
769         ../common/complex/conjf.c \
770         ../common/complex/conjl.c \
771         ../common/complex/cpow.c \
772         ../common/complex/cpowf.c \
773         ../common/complex/cpowl.c \
774         ../common/complex/cproj.c \
775         ../common/complex/cprojf.c \
776         ../common/complex/cprojl.c \
777         ../common/complex/creal.c \
778         ../common/complex/crealf.c \
779         ../common/complex/creall.c \
```

```
780         ../common/complex/csin.c \
781         ../common/complex/csinf.c \
782         ../common/complex/csinh.c \
783         ../common/complex/csinhf.c \
784         ../common/complex/csinhl.c \
785         ../common/complex/csinl.c \
786         ../common/complex/csqrt.c \
787         ../common/complex/csqrtf.c \
788         ../common/complex/csqrtl.c \
789         ../common/complex/ctan.c \
790         ../common/complex/ctanf.c \
791         ../common/complex/ctanh.c \
792         ../common/complex/ctanhf.c \
793         ../common/complex/ctanhl.c \
794         ../common/complex/ctanl.c \
795         ../common/complex/k_atan2.c \
796         ../common/complex/k_atan2l.c \
797         ../common/complex/k_cexp.c \
798         ../common/complex/k_cexpl.c \
799         ../common/complex/k_clog_r.c \
800         ../common/complex/k_clog_rl.c

802 SRCS_m9x_i386 = \
803         ../common/m9x/__fex_sse.c \
804         ../common/m9x/feprec.c \
805         ../common/m9x/__fex_i386.c

807 SRCS_m9x_i386_i386 = \
808         ../common/m9x/lroundf.c

810 SRCS_m9x_i386_amd64 = \
811         ../common/m9x/llrint.c \
812         ../common/m9x/llrintf.c \
813         ../common/m9x/llrintl.c \
814         ../common/m9x/nexttowardl.c \
815         ../common/m9x/remquo.c \
816         ../common/m9x/remquof.c \
817         ../common/m9x/round.c \
818         ../common/m9x/roundl.c \
819         ../common/m9x/scalbln.c \
820         ../common/m9x/scalblnf.c \
821         ../common/m9x/scalblnl.c \
822         ../common/m9x/trunc.c \
823         ../common/m9x/truncl.c

825 # sparc
826 SRCS_m9x_sparc_sparc = \
827         ../common/m9x/lrint.c \
828         ../common/m9x/lrintf.c \
829         ../common/m9x/lrintl.c \
830         ../common/m9x/lround.c \
831         ../common/m9x/lroundf.c \
832         ../common/m9x/lroundl.c

834 SRCS_m9x_sparc = \
835         ../common/m9x/__fex_sparc.c \
836         ../common/m9x/llrint.c \
837         ../common/m9x/llrintf.c \
838         ../common/m9x/llrintl.c \
839         ../common/m9x/nexttowardl.c \
840         ../common/m9x/remquo.c \
841         ../common/m9x/remquof.c \
842         ../common/m9x/remquol.c \
843         ../common/m9x/round.c \
844         ../common/m9x/roundl.c \
845         ../common/m9x/scalbln.c \
```



```
846         ../common/m9x/scalblnf.c \
847         ../common/m9x/scalblnl.c \
848         ../common/m9x/trunc.c \
849         ../common/m9x/truncl.c

851 SRCS_m9x = \
852         $(SRCS_m9x_$(MACH)) \
853         $(SRCS_m9x_sparc_$(TARGET_ARCH)) \
854         $(SRCS_m9x_i386_$(TARGET_ARCH)) \
855         ../common/m9x/__fex_hdlr.c \
856         ../common/m9x/__fex_sym.c \
857         ../common/m9x/fdim.c \
858         ../common/m9x/fdimf.c \
859         ../common/m9x/fdiml.c \
860         ../common/m9x/feexcept.c \
861         ../common/m9x/fenv.c \
862         ../common/m9x/feround.c \
863         ../common/m9x/fex_handler.c \
864         ../common/m9x/fex_log.c \
865         ../common/m9x/fma.c \
866         ../common/m9x/fmaf.c \
867         ../common/m9x/fmal.c \
868         ../common/m9x/fmax.c \
869         ../common/m9x/fmaxf.c \
870         ../common/m9x/fmaxl.c \
871         ../common/m9x/fmin.c \
872         ../common/m9x/fminf.c \
873         ../common/m9x/fminl.c \
874         ../common/m9x/frexp.c \
875         ../common/m9x/frexpf.c \
876         ../common/m9x/frexpl.c \
877         ../common/m9x/ldexp.c \
878         ../common/m9x/ldexpf.c \
879         ../common/m9x/ldexpl.c \
880         ../common/m9x/llround.c \
881         ../common/m9x/llroundf.c \
882         ../common/m9x/llroundl.c \
883         ../common/m9x/modf.c \
884         ../common/m9x/modff.c \
885         ../common/m9x/modfl.c \
886         ../common/m9x/nan.c \
887         ../common/m9x/nanf.c \
888         ../common/m9x/nanl.c \
889         ../common/m9x/nearbyint.c \
890         ../common/m9x/nearbyintf.c \
891         ../common/m9x/nearbyintl.c \
892         ../common/m9x/nexttoward.c \
893         ../common/m9x/nexttowardf.c \
894         ../common/m9x/roundf.c \
895         ../common/m9x/tgamma.c \
896         ../common/m9x/tgammaf.c \
897         ../common/m9x/tgammal.c \
898         ../common/m9x/truncf.c

900 SRCS_C_sparc = \
901         ../common/C/__tan.c \
902         ../common/C/_TBL_atan.c \
903         ../common/C/_TBL_exp2.c \
904         ../common/C/_TBL_log.c \
905         ../common/C/_TBL_log2.c \
906         ../common/C/_TBL_tan.c \
907         ../common/C/acos.c \
908         ../common/C/asin.c \
909         ../common/C/atan.c \
910         ../common/C/atan2.c \
911         ../common/C/ceil.c \
```

```
912         ../common/C/cos.c \
913         ../common/C/exp.c \
914         ../common/C/exp10.c \
915         ../common/C/exp2.c \
916         ../common/C/expm1.c \
917         ../common/C/floor.c \
918         ../common/C/fmod.c \
919         ../common/C/hypot.c \
920         ../common/C/ilogb.c \
921         ../common/C/isnan.c \
922         ../common/C/log.c \
923         ../common/C/log10.c \
924         ../common/C/log2.c \
925         ../common/C/pow.c \
926         ../common/C/remainder.c \
927         ../common/C/rint.c \
928         ../common/C/scalbn.c \
929         ../common/C/sin.c \
930         ../common/C/sincos.c \
931         ../common/C/tan.c

933 SRCS_i386_i386  = \
934         ../common/C/__libx_errno.c

936 SRCS_sparc_sparc = \
937         $(SRCS_i386_i386)

939 SRCS_sparc_sparcv9 = \
940         ../common/C/copysign.c \
941         ../common/C/fabs.c \
942         ../common/C/nextafter.c

944 SRCS_i386_amd64 = \
945         ../common/C/_TBL_atan.c \
946         ../common/C/_TBL_exp2.c \
947         ../common/C/_TBL_log.c \
948         ../common/C/_TBL_log2.c \
949         ../common/C/__tan.c \
950         ../common/C/_TBL_tan.c \
951         ../common/C/copysign.c \
952         ../common/C/exp.c \
953         ../common/C/fabs.c \
954         ../common/C/ilogb.c \
955         ../common/C/isnan.c \
956         ../common/C/nextafter.c \
957         ../common/C/rint.c \
958         ../common/C/scalbn.c \
959         ../common/C/acos.c \
960         ../common/C/asin.c \
961         ../common/C/atan.c \
962         ../common/C/atan2.c \
963         ../common/C/ceil.c \
964         ../common/C/cos.c \
965         ../common/C/exp10.c \
966         ../common/C/exp2.c \
967         ../common/C/expm1.c \
968         ../common/C/floor.c \
969         ../common/C/hypot.c \
970         ../common/C/log.c \
971         ../common/C/log10.c \
972         ../common/C/log2.c \
973         ../common/C/pow.c \
974         ../common/C/sin.c \
975         ../common/C/sincos.c \
976         ../common/C/tan.c
```



```
978 SRCS_C = \
979         $(SRCS_C_$(MACH)) \
980         $(SRCS_C_i386_$(TARGET_ARCH)) \
981         ../common/C/__cos.c \
982         ../common/C/__lgamma.c \
983         ../common/C/__rem_pio2.c \
984         ../common/C/__rem_pio2m.c \
985         ../common/C/__sin.c \
986         ../common/C/__sincos.c \
987         ../common/C/__xpg6.c \
988         ../common/C/_lib_version.c \
989         ../common/C/_SVID_error.c \
990         ../common/C/_TBL_ipio2.c \
991         ../common/C/_TBL_sin.c \
992         ../common/C/acosh.c \
993         ../common/C/asinh.c \
994         ../common/C/atan2pi.c \
995         ../common/C/atanh.c \
996         ../common/C/cbrt.c \
997         ../common/C/cosh.c \
998         ../common/C/erf.c \
999         ../common/C/gamma.c \
1000        ../common/C/gamma_r.c \
1001        ../common/C/j0.c \
1002        ../common/C/j1.c \
1003        ../common/C/jn.c \
1004        ../common/C/lgamma.c \
1005        ../common/C/lgamma_r.c \
1006        ../common/C/log1p.c \
1007        ../common/C/logb.c \
1008        ../common/C/matherr.c \
1009        ../common/C/scalb.c \
1010        ../common/C/signgam.c \
1011        ../common/C/significand.c \
1012        ../common/C/sincospi.c \
1013        ../common/C/sinh.c \
1014        ../common/C/sqrt.c \
1015        ../common/C/tanh.c

1017 SRCS    = \
1018        $(SRCS_Q_$(MACH)) \
1019        $(SRCS_LD_$(MACH)) \
1020        $(SRCS_R) \
1021        $(SRCS_complex) \
1022        $(SRCS_C)

1024 .KEEP_STATE:

1026 all:    $(LIBS)

1028 lint:   lintcheck
```

```
**********************************************************
    3962 Thu Oct  9 19:48:52 2014
new/usr/src/lib/libm/amd64/src/libm_inlines.h
review fixes for libm/amd64/src/libm_inlines.h
fix tmpd in copysign()
libm fixes from richlowe - richlowe.net/webrevs/il_keith
patch01 - 693 import Sun Devpro Math Library
review fixes for libm/amd64/src/libm_inlines.h
fix tmpd in copysign()
libm fixes from richlowe - richlowe.net/webrevs/il_keith
patch01 - 693 import Sun Devpro Math Library
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" 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 __inline__ float
__inline_sqrtf(float a)
{
        float ret;

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



```
}

extern __inline__ double
__inline_sqrt(double a)
{
        double ret;

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

extern __inline__ double
__ieee754_sqrt(double a)
{
        return (__inline_sqrt(a));
}

/*
 * 00 - 24 bits
 * 01 - reserved
 * 10 - 53 bits
 * 11 - 64 bits
 */
extern __inline__ int
__swapRP(int i)
{
        int ret;
        uint16_t cw;

        __asm__ __volatile__("fstcw %0\n\t" : "=m" (cw));

        ret = (cw >> 8) & 0x3;
        cw = (cw & 0xfcff) | ((i & 0x3) << 8);

        __asm__ __volatile__("fldcw %0\n\t" : : "m" (cw));

        return (ret);
}

/*
 * 00 - Round to nearest, with even preferred
 * 01 - Round down
 * 10 - Round up
 * 11 - Chop
 */
extern __inline__ enum fp_direction_type
__swap87RD(enum fp_direction_type i)
{
        int ret;
        uint16_t cw;

        __asm__ __volatile__("fstcw %0\n\t" : "=m" (cw));

        ret = (cw >> 10) & 0x3;
        cw = (cw & 0xf3ff) | ((i & 0x3) << 10);

        __asm__ __volatile__("fldcw %0\n\t" : : "m" (cw));

        return (ret);
}

extern __inline__ int
abs(int i)
{
        int ret;
        __asm__ __volatile__(
```

```
121             "movl    %1, %0\n\t"
122             "negl    %1\n\t"
123             "cmovnsl %1, %0\n\t"
124             : "=r" (ret), "+r" (i)
125             :
126             : "cc");
127         return (ret);
128 }

130 extern __inline__ double
131 copysign(double d1, double d2)
132 {
133         double tmpd;

135         __asm__ __volatile__(
136             "movd %3, %1\n\t"
137             "andpd %1, %0\n\t"
138             "andnpd %2, %1\n\t"
139             "orpd %1, %0\n\t"
140             : "+&x" (d1), "=&x" (tmpd)
141             : "x" (d2), "r" (0x7fffffffffffffff));

143         return (d1);
144 }

146 extern __inline__ double
147 fabs(double d)
148 {
149         double tmp;

151         __asm__ __volatile__(
152             "movd  %2, %1\n\t"
153             "andpd %1, %0"
154             : "+x" (d), "=&x" (tmp)
155             : "r" (0x7fffffffffffffff));

157         return (d);
158 }

160 extern __inline__ float
161 fabsf(float d)
162 {
163         __asm__ __volatile__(
164             "andpd %1, %0"
165             : "+x" (d)
166             : "x" (0x7fffffff));

168         return (d);
169 }

171 extern __inline__ int
172 finite(double d)
173 {
174         long ret = 0x7fffffffffffffff;
175         uint64_t tmp;

177         __asm__ __volatile__(
178             "movq %2, %1\n\t"
179             "andq %1, %0\n\t"
180             "movq $0x7ff0000000000000, %1\n\t"
181             "subq %1, %0\n\t"
182             "shrq $63, %0\n\t"
183             : "+r" (ret), "=r" (tmp)
184             : "x" (d)
185             : "cc");
```



```
187         return (ret);
188 }

190 extern __inline__ int
191 signbit(double d)
192 {
193         long ret;
194         __asm__ __volatile__(
195             "movmskpd %1, %0\n\t"
196             "andq     $1, %0\n\t"
197             : "=r" (ret)
198             : "x" (d)
199             : "cc");
200         return (ret);
201 }

203 extern __inline__ double
204 sqrt(double d)
205 {
206         return (__inline_sqrt(d));
207 }

209 extern __inline__ float
210 sqrtf(float f)
211 {
212         return (__inline_sqrtf(f));
213 }

215 #ifdef __cplusplus
216 }
217 #endif

219 #endif  /* __GNUC__ */

221 #endif /* _LIBM_INLINES_H */
```

```
**********************************************************
    5693 Thu Oct  9 19:48:52 2014
new/usr/src/lib/libm/common/C/__tan.c
libm - cstyle fixes
__tan.c
patch07 - removed dead code with mtsk.h
patch06 - libm: fixed compilation issues after updates
patch01 - 693 import Sun Devpro Math Library
libm - cstyle fixes
__tan.c
patch07 - removed dead code with mtsk.h
patch06 - libm: fixed compilation issues after updates
patch01 - 693 import Sun Devpro Math Library
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" 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.
 */

/* 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[];
static const double q[] = {
/* one  = */  1.0,
/*
 *                          2       2       -59.56
 * |sin(x) - pp1*x*(pp2+x *(pp3+x )| <= 2        for |x|<1/64
 */
/* pp1  = */  8.33326120969096230395312119298978359438478946686e-0003,
/* pp2  = */  1.20001038589438965215025680596868692381425944526e+0002,
/* pp3  = */ -2.00001730975089451192161504877731204032897949219e+0001,

/*
 *                      2      2        -56.19
 * |cos(x) - (1+qq1*x (qq2+x ))| <= 2        for |x|<=1/128
 */
/* qq1  = */  4.16665486385721928197511942926212213933467864990e-0002,
/* qq2  = */ -1.20000339921340035687080671777948737144470214844e+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 = */  3.71923358986516816929168705030406272271648049355e-0003,
/* t2 = */  6.02645120354857866118436504621058702468872070312e+0000,
/* t3 = */  2.42627327587398156083509093150496482849121093750e+0000,
/* t4 = */  2.44968983934252770851003333518747240304946899414e+0000,
/* t5 = */  6.07089252571767978849948121933266520500183105469e+0000,
/* t6 = */ -2.49403756995593761658369658107403665781021118164e+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 */


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

        t = one;
        hx = ((int *) &x)[HIWORD];
        ix = hx & 0x7fffffff;
        if (ix < 0x3fc40000) {          /* 0.15625 */
                if (ix < 0x3e400000) {  /* 2^-27 */
                        if ((i = (int) x) == 0)         /* generate inexact */
                                w = x;
                        t = y;
                } else {
                        z = x * x;
                        t = y + (((t1 * x) * z) * (t2 + z * (t3 + z))) *
                                ((t4 + z) * (t5 + z * (t6 + z)));
                        w = x + t;
                }
                if (k == 0)
                        return (w);
                /*
                 * Compute -1/(x+T) with great care
                 * Let r = -1/(x+T), rh = r chopped to 20 bits.
                 * Also let xh  = x+T chopped to 20 bits, xl = (x-xh)+T. Then
                 *   -1/(x+T)   = rh + (-1/(x+T)-rh) = rh + r*(1+rh*(x+T))
                 *              = rh + r*((1+rh*xh)+rh*xl).
                 */
                rh = r = -one / w;
                ((int *) &rh)[LOWORD] = 0;
                xh = w;
                ((int *) &xh)[LOWORD] = 0;
                xl = (x - xh) + t;
                return (rh + r * ((one + rh * xh) + rh * xl));
        }
        j = (ix + 0x4000) & 0x7fff8000;
        i = (j - 0x3fc40000) >> 15;
        ((int *) &t)[HIWORD] = j;
        if (hx > 0)
                x = y - (t - x);
        else
                x = -y - (t + x);
        a = _TBL_tan_hi[i];
        z = x * x;
        s = (pp1 * x) * (pp2 + z * (pp3 + z));  /* sin(x) */
        t = (qq1 * z) * (qq2 + z);              /* cos(x) - 1 */
        if (k == 0) {
                w = a * s;
                t = _TBL_tan_lo[i] + (s + a * w) / (one - (w - t));
                return (hx < 0 ? -a - t : a + t);
        } else {
                w = s + a * t;
                c = w + _TBL_tan_lo[i];
                t = a * s - t;
                /*
                 * Now try to compute [(1-T)/(a+c)] accurately
                 *
                 * Let r = 1/(a+c), rh = (1-T)*r chopped to 20 bits.
                 * Also let xh = a+c chopped to 20 bits, xl = (a-xh)+c. Then
                 *      (1-T)/(a+c) = rh + ((1-T)/(a+c)-rh)
                 *                  = rh + r*(1-T-rh*(a+c))
                 *                  = rh + r*((1-T-rh*xh)-rh*xl)
                 *                  = rh + r*(((1-rh*xh)-T)-rh*xl)
```



```
                 */
                r = one / (a + c);
                rh = (one - t) * r;
                ((int *) &rh)[LOWORD] = 0;
                xh = a + c;
                ((int *) &xh)[LOWORD] = 0;
                xl = (a - xh) + c;
                z = rh + r * (((one - rh * xh) - t) - rh * xl);
                return (hx >= 0 ? -z : z);
        }
}
```

```
**********************************************************
    4676 Thu Oct  9 19:48:52 2014
new/usr/src/lib/libm/common/C/acos.c
libm - more cstyle fixes
patch01 - 693 import Sun Devpro Math Library
libm - more cstyle fixes
patch01 - 693 import Sun Devpro Math Library
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" 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 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_synonyms.h"      /* __acos, __sqrt, __isnan */
#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 */
/* 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 */       -6.88238971605453293030e-01,    /* BFE6066C, 1B8D0159 */
/* qS4 */        7.70381505559019352791e-02     /* 3FB3B8C5, B12E9282 */
};
#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) {
        double z, p, q, r, w, s, c, df;
        int hx, ix;

        hx = ((int *) &x)[HIWORD];
        ix = hx & 0x7fffffff;
        if (ix >= 0x3ff00000) { /* |x| >= 1 */
                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))
#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 */
#endif
                else
                        return (_SVID_libm_err(x, x, 1));
        }
        if (ix < 0x3fe00000) {  /* |x| < 0.5 */
                if (ix <= 0x3c600000)
                        return (pio2_hi + pio2_lo);     /* if |x| < 2**-57 */
                z = x * x;
```

```
125                 p = z * (pS0 + z * (pS1 + z * (pS2 + z * (pS3 +
126                         z * (pS4 + z * pS5)))));
127                 q = one + z * (qS1 + z * (qS2 + z * (qS3 + z * qS4)));
128                 r = p / q;
129                 return (pio2_hi - (x - (pio2_lo - x * r)));
130         } else if (hx < 0) {
131                 /* x < -0.5 */
132                 z = (one + x) * 0.5;
133                 p = z * (pS0 + z * (pS1 + z * (pS2 + z * (pS3 +
134                         z * (pS4 + z * pS5)))));
135                 q = one + z * (qS1 + z * (qS2 + z * (qS3 + z * qS4)));
136                 s = sqrt(z);
137                 r = p / q;
138                 w = r * s - pio2_lo;
139                 return (pi - 2.0 * (s + w));
140         } else {
141                 /* x > 0.5 */
142                 z = (one - x) * 0.5;
143                 s = sqrt(z);
144                 df = s;
145                 ((int *) &df)[LOWORD] = 0;
146                 c = (z - df * df) / (s + df);
147                 p = z * (pS0 + z * (pS1 + z * (pS2 + z * (pS3 +
148                         z * (pS4 + z * pS5)))));
149                 q = one + z * (qS1 + z * (qS2 + z * (qS3 + z * qS4)));
150                 r = p / q;
151                 w = r * s + c;
152                 return (2.0 * (df + w));
153         }
154 }
```

```
**********************************************************
    2619 Thu Oct  9 19:48:52 2014
new/usr/src/lib/libm/common/C/acosh.c
libm - more cstyle fixes
patch01 - 693 import Sun Devpro Math Library
libm - more cstyle fixes
patch01 - 693 import Sun Devpro Math Library
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" 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 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_synonyms.h"      /* __acosh, __log, __log1p */
#include "libm_protos.h"        /* _SVID_libm_error */
#include "libm_macros.h"
#include <math.h>

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

double
```



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

        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));
                        /* assumes sparc-like QNaN */
#else
                        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) */
                        return (log(x) + ln2);
        } 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)));
        }
}
```

```
**********************************************************
    4878 Thu Oct  9 19:48:52 2014
new/usr/src/lib/libm/common/C/asin.c
libm - more cstyle fixes
In case of |x| < 2**-27 we are talking about x being almost 0.
The graph of arcsine curve shows that arcsin(0) is 0 so we will return 0 if x ==
or return x and generate inexact if x is close to 0.
patch07 - removed dead code with mtsk.h
patch06 - libm: fixed compilation issues after updates
patch01 - 693 import Sun Devpro Math Library
libm - more cstyle fixes
In case of |x| < 2**-27 we are talking about x being almost 0.
The graph of arcsine curve shows that arcsin(0) is 0 so we will return 0 if x ==
or return x and generate inexact if x is close to 0.
patch07 - removed dead code with mtsk.h
patch06 - libm: fixed compilation issues after updates
patch01 - 693 import Sun Devpro Math Library
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" 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 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_synonyms.h"      /* __asin, __sqrt, __isnan */
#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 */
/* 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 */
/* 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 */       -6.88238971605453293030e-01,    /* BFE6066C, 1B8D0159 */
/* qS4 */        7.70381505559019352791e-02     /* 3FB3B8C5, B12E9282 */
};
#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) {
        double t, w, p, q, c, r, s;
        int hx, ix, i;

        hx = ((int *) &x)[HIWORD];
        ix = hx & 0x7fffffff;
        if (ix >= 0x3ff00000) { /* |x| >= 1 */
                if (((ix - 0x3ff00000) | ((int *) &x)[LOWORD]) == 0)
```

```
115                         /* asin(1)=+-pi/2 with inexact */
116                         return (x * pio2_hi + x * pio2_lo);
117                 else if (isnan(x))
118 #if defined(FPADD_TRAPS_INCOMPLETE_ON_NAN)
119                         return (ix >= 0x7ff80000 ? x : (x - x) / (x - x));
120                         /* assumes sparc-like QNaN */
121 #else
122                         return (x - x) / (x - x);       /* asin(|x|>1) is NaN */
123 #endif
124                 else
125                         return (_SVID_libm_err(x, x, 2));
126         } else if (ix < 0x3fe00000) {   /* |x| < 0.5 */
127                 if (ix < 0x3e400000) {  /* if |x| < 2**-27 */
128                         if ((i = (int) x) == 0)
129                                 /* return x with inexact if x != 0 */
130                                 return (x);
131                 }
132                 t = x * x;
133                 p = t * (pS0 + t * (pS1 + t * (pS2 + t * (pS3 +
134                         t * (pS4 + t * pS5)))));
135                 q = one + t * (qS1 + t * (qS2 + t * (qS3 + t * qS4)));
136                 w = p / q;
137                 return (x + x * w);
138         }
139         /* 1 > |x| >= 0.5 */
140         w = one - fabs(x);
141         t = w * 0.5;
142         p = t * (pS0 + t * (pS1 + t * (pS2 + t * (pS3 + t * (pS4 + t * pS5)))));
143         q = one + t * (qS1 + t * (qS2 + t * (qS3 + t * qS4)));
144         s = sqrt(t);
145         if (ix >= 0x3FEF3333) { /* if |x| > 0.975 */
146                 w = p / q;
147                 t = pio2_hi - (2.0 * (s + s * w) - pio2_lo);
148         } else {
149                 w = s;
150                 ((int *) &w)[LOWORD] = 0;
151                 c = (t - w * w) / (s + w);
152                 r = p / q;
153                 p = 2.0 * s * r - (pio2_lo - 2.0 * c);
154                 q = pio4_hi - 2.0 * w;
155                 t = pio4_hi - (p - q);
156         }
157         return (hx > 0 ? t : -t);
158 }
```

```
**********************************************************
    2434 Thu Oct  9 19:48:52 2014
new/usr/src/lib/libm/common/C/asinh.c
libm - more cstyle fixes
patch01 - 693 import Sun Devpro Math Library
libm - more cstyle fixes
patch01 - 693 import Sun Devpro Math Library
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" 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 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_synonyms.h"      /* __asinh */
#include "libm_macros.h"
#include <math.h>

static const double xxx[] = {
/* one */       1.00000000000000000000e+00,     /* 3FF00000, 00000000 */
/* ln2 */       6.93147180559945286227e-01,     /* 3FE62E42, FEFA39EF */
/* huge */      1.00000000000000000000e+300
};
#define one     xxx[0]
#define ln2     xxx[1]
#define huge    xxx[2]
```



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

        hx = ((int *) &x)[HIWORD];
        ix = hx & 0x7fffffff;
        if (ix >= 0x7ff00000)
#if defined(FPADD_TRAPS_INCOMPLETE_ON_NAN)
                return (ix >= 0x7ff80000 ? x : x + x);
                /* assumes sparc-like QNaN */
#else
                return (x + x); /* x is inf or NaN */
#endif
        if (ix < 0x3e300000) {  /* |x|<2**-28 */
                if (huge + x > one)
                        return (x);     /* return x inexact except 0 */
        }
        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)));
        }
        return (hx > 0 ? w : -w);
}
```

```
**********************************************************
    2088 Thu Oct  9 19:48:52 2014
new/usr/src/lib/libm/common/C/atanh.c
libm - more cstyle fixes
patch01 - 693 import Sun Devpro Math Library
libm - more cstyle fixes
patch01 - 693 import Sun Devpro Math Library
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" 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 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_synonyms.h"
#include "libm_protos.h"
#include <math.h>

double
```



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

        if (isnan(x))
                return (x * x);         /* switched from x + x for Cheetah */
        t = fabs(x);
        if (t > 1.0)
                return (_SVID_libm_err(x, x, 30));      /* sNaN */
        if (t == 1.0)
                return (_SVID_libm_err(x, x, 31));      /* x/0; */
        t = t / (1.0 - t);
        return (copysign(0.5, x) * log1p(t + t));
}
```

```
**********************************************************
    2452 Thu Oct  9 19:48:53 2014
new/usr/src/lib/libm/common/C/cosh.c
libm - more cstyle fixes
patch01 - 693 import Sun Devpro Math Library
libm - more cstyle fixes
patch01 - 693 import Sun Devpro Math Library
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" 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.
 */

#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,
        ln2hi = 6.93147180369123816490e-01,
        ln2lo = 1.90821492927058770002e-10,
        lnovft = 7.09782712893383973096e+02;

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

        w = fabs(x);
        if (!finite(w))
                return (w * w);
        if (w < 0.3465) {
                t = expm1(w);
                w = 1.0 + t;
                if (w != 1.0)
                        w = 1.0 + (t * t) / (w + w);
                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;
                if (w >= ln2)
                        return (_SVID_libm_err(x, x, 5));
                else
                        return (scalbn(exp(w), 1023));
        }
}
```

```
**********************************************************
   13875 Thu Oct  9 19:48:53 2014
new/usr/src/lib/libm/common/C/erf.c
libm - more cstyle fixes
patch01 - 693 import Sun Devpro Math Library
libm - more cstyle fixes
patch01 - 693 import Sun Devpro Math Library
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" 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 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_synonyms.h"      /* __erf, __erfc, __exp */
#include "libm_macros.h"
#include <math.h>

static const double xxx[] = {
/* 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 */
/*
 * 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 */
/*
 * Coefficients for approximation to  erf  in [0.84375,1.25]
 */
/* pa0 */       -2.36211856075265944077e-03,    /* BF6359B8, BEF77538 */
/* pa1 */        4.14855118683748331666e-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 */        7.18285644141962662868e-02,    /* 3FB2635C, D99FE9A7 */
/* qa4 */        1.26171219808761642112e-01,    /* 3FC02660, E763351F */
/* qa5 */        1.36370839120290507362e-02,    /* 3F8BEDC2, 6B51DD1C */
/* qa6 */        1.19844998467991074170e-02,    /* 3F888B54, 5735151D */
/*
 * 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 */
/*
```



```
 * 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]
/*
 * 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]
/*
 * 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]
/*
 * 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]
/*
 * 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) {
        int hx, ix, i;
        double R, S, P, Q, s, y, z, r;

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

        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));
                        return (x + efx * x);
                }
                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))));
                y = r / s;
                return (x + x * y);
        }
        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)))));
                if (hx >= 0)
                        return (erx + P / Q);
                else
                        return (-erx - P / Q);
        }
        if (ix >= 0x40180000) { /* inf > |x| >= 6 */
```



```
                if (hx >= 0)
                        return (one - tiny);
                else
                        return (tiny - one);
        }
        x = fabs(x);
        s = one / (x * x);
        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)))))));
        } 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))))));
        }
        z = x;
        ((int *) &z)[LOWORD] = 0;
        r = exp(-z * z - 0.5625) * exp((z - x) * (z + x) + R / S);
        if (hx >= 0)
                return (one - r / x);
        else
                return (r / x - one);
}

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

        hx = ((int *) &x)[HIWORD];
        ix = hx & 0x7fffffff;
        if (ix >= 0x7ff00000) { /* erfc(nan)=nan */
#if defined(FPADD_TRAPS_INCOMPLETE_ON_NAN)
                if (ix >= 0x7ff80000)           /* assumes sparc-like QNaN */
                        return (x);
#endif
                /* erfc(+-inf)=0,2 */
                return ((double) (((unsigned) hx >> 31) << 1) + one / x);
        }

        if (ix < 0x3feb0000) {  /* |x| < 0.84375 */
                if (ix < 0x3c700000)    /* |x| < 2**-56 */
                        return (one - x);
                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))));
                y = r / s;
                if (hx < 0x3fd00000) {  /* x < 1/4 */
                        return (one - (x + x * y));
                } else {
                        r = x * y;
                        r += (x - half);
                        return (half - r);
                }
        }
        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)))));
                if (hx >= 0) {
```

```
389                         z = one - erx;
390                         return (z - P / Q);
391                 } else {
392                         z = erx + P / Q;
393                         return (one + z);
394                 }
395         }
396         if (ix < 0x403c0000) {  /* |x|<28 */
397                 x = fabs(x);
398                 s = one / (x * x);
399                 if (ix < 0x4006DB6D) {  /* |x| < 1/.35 ~ 2.857143 */
400                         R = ra0 + s * (ra1 + s * (ra2 + s * (ra3 + s * (ra4 +
401                                 s * (ra5 + s * (ra6 + s * ra7))))));
402                         S = one + s * (sa1 + s * (sa2 + s * (sa3 + s * (sa4 +
403                                 s * (sa5 + s * (sa6 + s * (sa7 + s * sa8)))))));
404                 } else {
405                         /* |x| >= 1/.35 ~ 2.857143 */
406                         if (hx < 0 && ix >= 0x40180000)
407                                 return (two - tiny);    /* x < -6 */

409                         R = rb0 + s * (rb1 + s * (rb2 + s * (rb3 + s * (rb4 +
410                                 s * (rb5 + s * rb6)))));
411                         S = one + s * (sb1 + s * (sb2 + s * (sb3 + s * (sb4 +
412                                 s * (sb5 + s * (sb6 + s * sb7))))));
413                 }
414                 z = x;
415                 ((int *) &z)[LOWORD] = 0;
416                 r = exp(-z * z - 0.5625) * exp((z - x) * (z + x) + R / S);
417                 if (hx > 0)
418                         return (r / x);
419                 else
420                         return (two - r / x);
421         } else {
422                 if (hx > 0)
423                         return (tiny * tiny);
424                 else
425                         return (two - tiny);
426         }
427 }
```

```
**********************************************************
    8517 Thu Oct  9 19:48:53 2014
new/usr/src/lib/libm/common/C/expm1.c
libm - more cstyle fixes
common/C/expm1.c: cstyle
nextafter.c
log1p.c
It is safe to initialize c with 0 at the beginning.
We will use 'c' only if (k != 0).
libm/common/C/expm1.c
For |x| <= 0.5 ln2, (k == 0) we don't need to compute correction (c).
We can initialize c manually with 0, to fix compiler warning.
patch07 - removed dead code with mtsk.h
patch06 - libm: fixed compilation issues after updates
patch01 - 693 import Sun Devpro Math Library
libm - more cstyle fixes
common/C/expm1.c: cstyle
nextafter.c
log1p.c
It is safe to initialize c with 0 at the beginning.
We will use 'c' only if (k != 0).
libm/common/C/expm1.c
For |x| <= 0.5 ln2, (k == 0) we don't need to compute correction (c).
We can initialize c manually with 0, to fix compiler warning.
patch07 - removed dead code with mtsk.h
patch06 - libm: fixed compilation issues after updates
patch01 - 693 import Sun Devpro Math Library
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" 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 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  =  -1.66666666666667384E-2,
 *              Q2  =   3.96825396813703658 73E-4,
 *              Q3  =  -9.92063447334359873 57E-6,
 *              Q4  =   2.50513614208085170 02E-7,
 *              Q5  =  -6.28435056823826171 02E-9;
 *      (where z=r*r, and the values of Q1 to Q5 are listed below)
 *      with error bounded by
 *          |                    5           |     -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_synonyms.h"      /* __expm1 */
#include "libm_macros.h"
#include <math.h>

static const double xxx[] = {
/* 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 */
/* Q2 */                1.58730158725481460165e-03,    /* 3F5A01A0 19FE5585 */
/* Q3 */               -7.93650757867487942473e-05,    /* BF14CE19 9EAADBB7 */
/* Q4 */                4.00821782732936239552e-06,    /* 3ED0CFCA 86E65239 */
/* Q5 */               -2.01099218183624371326e-07     /* BE8AFDB7 6E09C32D */
};
#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) {
        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 */
        xsb = hx & 0x80000000;                  /* sign bit of x */
        if (xsb == 0)
```



```
                y = x;
        else
                y = -x;                         /* y = |x| */
        hx &= 0x7fffffff;                       /* high word of |x| */

        /* 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)
                                        return (x * x); /* + -> * for Cheetah */
                                else
                                        /* exp(+-inf)={inf,-1} */
                                        return (xsb == 0 ? x : -1.0);
                        }
                        if (x > o_threshold)
                                return (huge * huge);   /* overflow */
                }
                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));
                        t = k;
                        hi = x - t * ln2_hi;    /* t*ln2_hi is exact here */
                        lo = t * ln2_lo;
                }
                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
                /* |x| <= 0.5 ln2 */
                k = 0;

        /* 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 */
                return (x - (x * e - hxs));
        else {          /* |x| > 0.5 ln2 */
```

```
237                 e = (x * (e - c) - c);
238                 e -= hxs;
239                 if (k == -1)
240                         return (0.5 * (x - e) - 0.5);
241                 if (k == 1) {
242                         if (x < -0.25)
243                                 return (-2.0 * (e - (x + 0.5)));
244                         else
245                                 return (one + 2.0 * (x - e));
246                 }
247                 if (k <= -2 || k > 56) {        /* suffice to return exp(x)-1 */
248                         y = one - (e - x);
249                         ((int *) &y)[HIWORD] += k << 20;
250                         return (y - one);
251                 }
252                 t = one;
253                 if (k < 20) {
254                         ((int *) &t)[HIWORD] = 0x3ff00000 - (0x200000 >> k);
255                                                         /* t = 1 - 2^-k */
256                         y = t - (e - x);
257                         ((int *) &y)[HIWORD] += k << 20;
258                 } else {
259                         ((int *) &t)[HIWORD] = (0x3ff - k) << 20; /* 2^-k */
260                         y = x - (e + t);
261                         y += one;
262                         ((int *) &y)[HIWORD] += k << 20;
263                 }
264         }
265         return (y);
266 }
```

```
**********************************************************
    1205 Thu Oct  9 19:48:53 2014
new/usr/src/lib/libm/common/C/fabs.c
libm - more cstyle fixes
patch01 - 693 import Sun Devpro Math Library
libm - more cstyle fixes
patch01 - 693 import Sun Devpro Math Library
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" 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 fabs = __fabs

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

double
fabs(double x) {
        int *px = (int *) &x;

        px[HIWORD] &= ~0x80000000;
        return (x);
}
```

```
**********************************************************
    8824 Thu Oct  9 19:48:53 2014
new/usr/src/lib/libm/common/C/j0.c
libm - more cstyle fixes
patch01 - 693 import Sun Devpro Math Library
libm - more cstyle fixes
patch01 - 693 import Sun Devpro Math Library
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" 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.
 */

/*
 * 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_synonyms.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 = 5.64189583547756286948079451560772585844le-0001,
tpi     = 0.63661977236758134307553505349005744 8;

static GENERIC pzero(GENERIC), qzero(GENERIC);
```



```
static const GENERIC r0[4] = {  /* [1.e-5, 1.28] */
        -2.50000000000000362213188089483047675553 7e-0001,
        1.09559754733483026323443385593237535330 3e-0002,
        -1.81973475046332092179918725898709808769 7e-0004,
        9.97700194680613165754421250106989393084 6e-0007,
};
static const GENERIC s0[4] = {  /* [1.e-5, 1.28] */
        1.0,
        1.86760981066295016996678236058819967374 1e-0002,
        1.59038920618156549087843082770697207420 8e-0004,
        6.52086738674258363237552014771449952272 1e-0007,
};
static const GENERIC r1[9] = {  /* [1.28,8] */
        9.99999999999999942156495584397047660949e-0001,
        -2.38988772273131913047683983690814373128 1e-0001,
        1.29335947613893902779127039343949364057 0e-0002,
        -2.77098564234314012216885240022856336408 2e-0004,
        2.90524157577206767808673838916962521891 2e-0006,
        -1.63684635626405259796904200926504325127 9e-0008,
        5.07230616072488477508543105905261173782 7e-0011,
        -8.18706073068406682422891477514653613911 2e-0014,
        5.42221932695994986395429786072372342384 2e-0017,
};
static const GENERIC s1[9] = {  /* [1.28,8] */
        1.0,
        1.10112277268680770276210474193207622834 9e-0002,
        6.14016931064164922341142776466914397822 8e-0005,
        2.29203587751515209797694611929321570525 0e-0007,
        6.35691042650464433455883203636221958378 9e-0010,
        1.36662632690021955504509699955394889140 1e-0012,
        2.28039958686673952289183798556048118008 8e-0015,
        2.80155982064893966527049252000483661118 7e-0018,
        2.07310108832034915976441026146635073296 8e-0021,
};

GENERIC
j0(GENERIC x) {
        GENERIC z, s, c, ss, cc, r, u, v, ox;
        int i;

        if (isnan(x))
                return (x*x);   /* + -> * for Cheetah */
        ox = x;
        x = fabs(x);
        if (x > 8.0) {
                if (!finite(x))
                        return (zero);
                s = sin(x);
                c = cos(x);
        /*
         * 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.
         */
                if (x > 8.9e307) {      /* x+x may overflow */
                        ss = s-c;
                        cc = s+c;
                } else if (signbit(s) != signbit(c)) {
                        ss = s - c;
```

```
125                         cc = -cos(x+x)/ss;
126                 } else {
127                         cc = s + c;
128                         ss = -cos(x+x)/cc;
129                 }
130         /*
131          * j0(x) = 1/sqrt(pi) * (P(0,x)*cc - Q(0,x)*ss) / sqrt(x)
132          * y0(x) = 1/sqrt(pi) * (P(0,x)*ss + Q(0,x)*cc) / sqrt(x)
133          */
134                 if (x > 1.0e40) z = (invsqrtpi*cc)/sqrt(x);
135                 else {
136                     u = pzero(x); v = qzero(x);
137                     z = invsqrtpi*(u*cc-v*ss)/sqrt(x);
138                 }
139         /* force to pass SVR4 even the result is wrong (sign) */
140                 if (x > X_TLOSS)
141                     return (_SVID_libm_err(ox, z, 34));
142                 else
143                     return (z);
144         }
145         if (x <= small) {
146             if (x <= tiny)
147                         return (one-x);
148             else
149                         return (one-x*x*0.25);
150         }
151         z = x*x;
152         if (x <= 1.28) {
153             r =  r0[0]+z*(r0[1]+z*(r0[2]+z*r0[3]));
154             s =  s0[0]+z*(s0[1]+z*(s0[2]+z*s0[3]));
155             return (one + z*(r/s));
156         } else {
157             for (r = r1[8], s = s1[8], i = 7; i >= 0; i--) {
158                 r = r*z + r1[i];
159                 s = s*z + s1[i];
160             }
161             return (r/s);
162         }
163 }

165 static const GENERIC u0[13] = {
166         -7.38042951086872252675472302070431764l941e-0002,
167         1.77260710268486992430145966304987429481 4e-0001,
168         -1.52437066654271382860407809097079935 6306e-0002,
169         4.65081910069389175714377155762992459 1915e-0004,
170         -7.12576887233952897503631610871823994 6022e-0006,
171         6.41101700165610459832756500477151525 7146e-0008,
172         -3.69427515743303255302124681237925878 1665e-0010,
173         1.43436454420626662425282088964845263842e-0012,
174         -3.85206473185993645589503628687413989 6861e-0015,
175         7.18205289972613838173994588191487457 9696e-0018,
176         -9.06055657461967756732374119407979798 7200e-0021,
177         7.12443546740886051526555221731123051 1455e-0024,
178         -2.70972677463639761532881312171543204 4771e-0027,
179 };
180 static const GENERIC v0[5] = {
181         1.0,
182         4.67867893151254900258770247734921488 6475e-0003,
183         9.48682895552994853482280082949756517 8985e-0006,
184         1.00149592915886164665901084413668245 4906e-0008,
185         4.72533811625602166020444323568535859 3611e-0012,
186 };

188 GENERIC
189 y0(GENERIC x) {
190         GENERIC z, /* d, */ s, c, ss, cc, u, v;
```



```
191         int i;

193         if (isnan(x))
194                 return (x*x);   /* + -> * for Cheetah */
195         if (x <= zero) {
196                 if (x == zero)
197                     /* d= -one/(x-x); */
198                     return (_SVID_libm_err(x, x, 8));
199                 else
200                     /* d = zero/(x-x); */
201                     return (_SVID_libm_err(x, x, 9));
202         }
203         if (x > 8.0) {
204                 if (!finite(x))
205                         return (zero);
206                 s = sin(x);
207                 c = cos(x);
208         /*
209          * j0(x) = sqrt(2/(pi*x))*(p0(x)*cos(x0)-q0(x)*sin(x0))
210          * where x0 = x-pi/4
211          *      Better formula:
212          *              cos(x0) = cos(x)cos(pi/4)+sin(x)sin(pi/4)
213          *                      = 1/sqrt(2) * (cos(x) + sin(x))
214          *              sin(x0) = sin(x)cos(pi/4)-cos(x)sin(pi/4)
215          *                      = 1/sqrt(2) * (sin(x) - cos(x))
216          * To avoid cancellation, use
217          *              sin(x) +- cos(x) = -cos(2x)/(sin(x) -+ cos(x))
218          * to compute the worse one.
219          */
220                 if (x > 8.9e307) {      /* x+x may overflow */
221                         ss = s-c;
222                         cc = s+c;
223                 } else if (signbit(s) != signbit(c)) {
224                         ss = s - c;
225                         cc = -cos(x+x)/ss;
226                 } else {
227                         cc = s + c;
228                         ss = -cos(x+x)/cc;
229                 }
230         /*
231          * j0(x) = 1/sqrt(pi*x) * (P(0,x)*cc - Q(0,x)*ss)
232          * y0(x) = 1/sqrt(pi*x) * (P(0,x)*ss + Q(0,x)*cc)
233          */
234                 if (x > 1.0e40)
235                         z = (invsqrtpi*ss)/sqrt(x);
236                 else
237                     z =  invsqrtpi*(pzero(x)*ss+qzero(x)*cc)/sqrt(x);
238                 if (x > X_TLOSS)
239                     return (_SVID_libm_err(x, z, 35));
240                 else
241                     return (z);

243         }
244         if (x <= tiny) {
245             return (u0[0] + tpi*log(x));
246         }
247         z = x*x;
248         for (u = u0[12], i = 11; i >= 0; i--) u = u*z + u0[i];
249         v = v0[0]+z*(v0[1]+z*(v0[2]+z*(v0[3]+z*v0[4])));
250         return (u/v + tpi*(j0(x)*log(x)));
251 }

253 static const GENERIC pr[7] = {  /* [8 -- inf]  pzero 6550 */
254         .48613441833860527213912384 47e5,
255         .13776625494071122781334389 45e6,
256         .12224663640882897318691140 04e6,
```

```
257         .41070700843151761355833533374e5,
258         .50260738018606371258890039915e4,
259         .17831936591254796545415424l9e3,
260         .88010344055383421691677564e0,
261 };
262 static const GENERIC ps[7] = {  /* [8 -- inf] pzero 6550 */
263         .48613441833860527214140370 58e5,
264         .13781966326303846704775826 99e6,
265         .12239671853410065427489367 87e6,
266         .41201502437953536399958626 17e5,
267         .50682711810535463924901843 53e4,
268         .18298179054727699605356716 64e3,
269         1.0,
270 };
271 static const GENERIC huge    = 1.0e10;

273 static GENERIC
274 pzero(GENERIC x) {
275         GENERIC s, r, t, z;
276         int i;
277         if (x > huge)
278                 return (one);
279         t = eight/x; z = t*t;
280         r = pr[5]+z*pr[6];
281         s = ps[5]+z;
282         for (i = 4; i >= 0; i--) {
283             r = r*z + pr[i];
284             s = s*z + ps[i];
285         }
286         return (r/s);
287 }

289 static const GENERIC qr[7] = {  /* [8 -- inf]  qzero 6950 */
290         -.17312109957010685391856119 51e3,
291         -.55225591659361669612352406 13e3,
292         -.56049356066373465906145296 13e3,
293         -.22004303002260093794773650 11e3,
294         -.32386935537564884977129674 6e2,
295         -.14294979207907956223499258 e1,
296         -.83469037410238498815891 8e-2,
297 };
298 static const GENERIC qs[7] = {  /* [8 -- inf] qzero 6950 */
299         .11079750372486838653267096 45e5,
300         .35445816806270826746514718 73e5,
301         .36191189379183941321790190 59e5,
302         .14398955635653980074714858 22e5,
303         .21902770233443639559302262 34e4,
304         .10669515702040798613750168 2e3,
305         1.0,
306 };

308 static GENERIC
309 qzero(GENERIC x) {
310         GENERIC s, r, t, z;
311         int i;
312         if (x > huge)
313                 return (-0.125/x);
314         t = eight/x; z = t*t;
315         r = qr[5]+z*qr[6];
316         s = qs[5]+z;
317         for (i = 4; i >= 0; i--) {
318             r = r*z + qr[i];
319             s = s*z + qs[i];
320         }
321         return (t*(r/s));
322 }
```

```
**********************************************************
    9194 Thu Oct  9 19:48:53 2014
new/usr/src/lib/libm/common/C/j1.c
libm - more cstyle fixes
common/C/j1.c: cstyle
patch01 - 693 import Sun Devpro Math Library
libm - more cstyle fixes
common/C/j1.c: cstyle
patch01 - 693 import Sun Devpro Math Library
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" 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.
 */

/*
 * 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_synonyms.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 = 5.64189583547756286948079451560772585844le-0001,
tpi     = 0.63661977236758134307553505349005744 8;
```



```
static GENERIC pone(GENERIC), qone(GENERIC);
static const GENERIC r0[4] = {
        -6.250000000000002203053200981413218949548e-0002,
        1.600998455640072901321605101981501263762e-0003,
        -1.963888159483137585525118843901626864930e-0005,
        8.263917341093549759781339713418201620998e-0008,
};
static const GENERIC s0[7] = {
        1.0e0,
        1.605069137643004242395356851797873766927e-0002,
        1.149454623251299996428500249509098499383e-0004,
        3.849701673735260970379681807910852327825e-0007,
};
static const GENERIC r1[12] = {
        4.999999999999999955174088943404854717 24e-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,
};
static const GENERIC s1[5] = {
        1.0e0,
        4.923499437590484879081138588998986303306e-0003,
        1.054389489212184156499666953501976688452e-0005,
        1.180768373106166527048240364872043816050e-0008,
        5.942665743476099355323245707680648588540e-0012,
};

GENERIC
j1(GENERIC x) {
        GENERIC z, d, s, c, ss, cc, r;
        int i, sgn;

        if (!finite(x))
                return (one/x);
        sgn = signbit(x);
        x = fabs(x);
        if (x > 8.00) {
                s = sin(x);
                c = cos(x);
        /*
         * 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.
         */
                if (x > 8.9e307) {      /* x+x may overflow */
                        ss = -s-c;
                        cc =  s-c;
                } else if (signbit(s) != signbit(c)) {
                        cc = s - c;
                        ss = cos(x+x)/cc;
                } else {
```

```
123                         ss = -s-c;
124                         cc = cos(x+x)/ss;
125                 }
126         /*
127          * j1(x) = 1/sqrt(pi*x) * (P(1,x)*cc - Q(1,x)*ss)
128          * y1(x) = 1/sqrt(pi*x) * (P(1,x)*ss + Q(1,x)*cc)
129          */
130                 if (x > 1.0e40)
131                     d = (invsqrtpi*cc)/sqrt(x);
132                 else
133                         d =  invsqrtpi*(pone(x)*cc-qone(x)*ss)/sqrt(x);
134
135                 if (x > X_TLOSS) {
136                     if (sgn != 0) { d = -d; x = -x; }
137                         return (_SVID_libm_err(x, d, 36));
138                 } else
139                     if (sgn == 0)
140                                 return (d);
141                         else
142                                 return (-d);
143         }
144         if (x <= small) {
145                 if (x <= tiny)
146                         d = 0.5*x;
147                 else
148                         d =  x*(0.5-x*x*0.125);
149                 if (sgn == 0)
150                         return (d);
151                 else
152                         return (-d);
153         }
154         z = x*x;
155         if (x < 1.28) {
156             r = r0[3];
157             s = s0[3];
158             for (i = 2; i >= 0; i--) {
159                 r = r*z + r0[i];
160                 s = s*z + s0[i];
161             }
162             d = x*0.5+x*(z*(r/s));
163         } else {
164             r = r1[11];
165             for (i = 10; i >= 0; i--) r = r*z + r1[i];
166             s = s1[0]+z*(s1[1]+z*(s1[2]+z*(s1[3]+z*s1[4])));
167             d = x*(r/s);
168         }
169         if (sgn == 0)
170                 return (d);
171         else
172                 return (-d);
173 }
174
175 static const GENERIC u0[4] = {
176         -1.960570906462389461018983259589655961560e-0001,
177         4.931824118350661953459180060007970291139e-0002,
178         -1.626975871565393656845930125424683008677e-0003,
179         1.359657517926394132692884168082224258360e-0005,
180 };
181 static const GENERIC v0[5] = {
182         1.0e0,
183         2.565807214838390835108224713630901653793e-0002,
184         3.374175208978404268650522752520906231508e-0004,
185         2.840368571306070719539936935220728843177e-0006,
186         1.396387402048998277638900944415752207592e-0008,
187 };
188 static const GENERIC u1[12] = {
```



```
189         -1.960570906462389473336339614647555351626e-0001,
190         5.336268030335074494231369159933012844735e-0002,
191         -2.684137504382748094149184541866332033280e-0003,
192         5.737671618979185736981543498580051903060e-0005,
193         -6.642696350686335339171171785557663224892e-0007,
194         4.692417922568160354012347591960362101664e-0009,
195         -2.161728635907789319335231338621412258355e-0011,
196         6.727353419738316107197644431844194668702e-0014,
197         -1.427502986803861372125234355906790573422e-0016,
198         2.020392498726806769468143219616642940371e-0019,
199         -1.761371948595104156753045457888272716340e-0022,
200         7.352828391941157905175042420249225115816e-0026,
201 };
202 static const GENERIC v1[5] = {
203         1.0e0,
204         5.029187436727947764916247076102283399442e-0003,
205         1.102693095808242775074856548927801750627e-0005,
206         1.268035774543174837829534603830227216291e-0008,
207         6.579416271766610825192542295821308730206e-0012,
208 };
209
210
211 GENERIC
212 y1(GENERIC x) {
213         GENERIC z, d, s, c, ss, cc, u, v;
214         int i;
215
216         if (isnan(x))
217                 return (x*x);   /* + -> * for Cheetah */
218         if (x <= zero) {
219                 if (x == zero)
220                     /* return -one/zero;  */
221                     return (_SVID_libm_err(x, x, 10));
222                 else
223                     /* return zero/zero; */
224                     return (_SVID_libm_err(x, x, 11));
225         }
226         if (x > 8.0) {
227                 if (!finite(x))
228                         return (zero);
229                 s = sin(x);
230                 c = cos(x);
231         /*
232          * j1(x) = sqrt(2/(pi*x))*(p1(x)*cos(x0)-q1(x)*sin(x0))
233          * where x0 = x-3pi/4
234          *      Better formula:
235          *              cos(x0) = cos(x)cos(3pi/4)+sin(x)sin(3pi/4)
236          *                      =  1/sqrt(2) * (sin(x) - cos(x))
237          *              sin(x0) = sin(x)cos(3pi/4)-cos(x)sin(3pi/4)
238          *                      = -1/sqrt(2) * (cos(x) + sin(x))
239          * To avoid cancellation, use
240          *              sin(x) +- cos(x) = -cos(2x)/(sin(x) -+ cos(x))
241          * to compute the worse one.
242          */
243                 if (x > 8.9e307) {      /* x+x may overflow */
244                         ss = -s-c;
245                         cc =  s-c;
246                 } else if (signbit(s) != signbit(c)) {
247                         cc = s - c;
248                         ss = cos(x+x)/cc;
249                 } else {
250                         ss = -s-c;
251                         cc = cos(x+x)/ss;
252                 }
253         /*
254          * j1(x) = 1/sqrt(pi*x) * (P(1,x)*cc - Q(1,x)*ss)
```

```
255          * y1(x) = 1/sqrt(pi*x) * (P(1,x)*ss + Q(1,x)*cc)
256          */
257                 if (x > 1.0e91)
258                     d =  (invsqrtpi*ss)/sqrt(x);
259                 else
260                         d = invsqrtpi*(pone(x)*ss+qone(x)*cc)/sqrt(x);

262                 if (x > X_TLOSS)
263                         return (_SVID_libm_err(x, d, 37));
264                 else
265                         return (d);
266         }
267                 if (x <= tiny) {
268                         return (-tpi/x);
269                 }
270         z = x*x;
271         if (x < 1.28) {
272             u = u0[3]; v = v0[3]+z*v0[4];
273             for (i = 2; i >= 0; i--) {
274                 u = u*z + u0[i];
275                 v = v*z + v0[i];
276             }
277         } else {
278             for (u = u1[11], i = 10; i >= 0; i--) u = u*z+u1[i];
279             v = v1[0]+z*(v1[1]+z*(v1[2]+z*(v1[3]+z*v1[4])));
280         }
281         return (x*(u/v) + tpi*(j1(x)*log(x)-one/x));
282 }

284 static const GENERIC pr0[6] = {
285         -.44357578167941278571147207 94e7,
286         -.99422465050776411956583778 99e7,
287         -.66033732483649391092552454 34e7,
288         -.15235293511811373832551057 22e7,
289         -.10982405543459346727374131 39e6,
290         -.16116166443246101164774128 98e4,
291 };
292 static const GENERIC ps0[6] = {
293         -.44357578167941278568280169 62e7,
294         -.99341243899345856589675563 09e7,
295         -.65853394797230870728269150 69e7,
296         -.15118095066341608816445465 58e7,
297         -.10726385991103820119030638 67e6,
298         -.14550094401904961824535650 68e4,
299 };
300 static const GENERIC huge    = 1.0e10;

302 static GENERIC
303 pone(GENERIC x) {
304         GENERIC s, r, t, z;
305         int i;
306                 /* assume x > 8 */
307         if (x > huge)
308                 return (one);

310         t = 8.0/x; z = t*t;
311         r = pr0[5]; s = ps0[5]+z;
312         for (i = 4; i >= 0; i--) {
313                 r = z*r + pr0[i];
314                 s = z*s + ps0[i];
315         }
316         return (r/s);
317 }


320 static const GENERIC qr0[6] = {
```



```
321         0.33220913409857223518597044 42e5,
322         0.85145160675335701965550011 71e5,
323         0.66178836581270835179399921 66e5,
324         0.18494262873223866796520098 19e5,
325         0.17063754290207680020612835 46e4,
326         0.35265133846636032185921755 80e2,
327 };
328 static const GENERIC qs0[6] = {
329         0.70871281941028743573775024 72e6,
330         0.18194580422439972989245538 39e7,
331         0.14194606696037208928557552 53e7,
332         0.40029443582266975117086108 13e6,
333         0.37890229745772202641429522 56e5,
334         0.86383677696049909674755171 83e3,
335 };

337 static GENERIC
338 qone(GENERIC x) {
339         GENERIC s, r, t, z;
340         int i;
341         if (x > huge)
342                 return (0.375/x);

344         t = 8.0/x; z = t*t;
345                 /* assume x > 8 */
346         r = qr0[5]; s = qs0[5]+z;
347         for (i = 4; i >= 0; i--) {
348                 r = z*r + qr0[i];
349                 s = z*s + qs0[i];
350         }
351         return (t*(r/s));
352 }
```

```
**********************************************************
    7423 Thu Oct  9 19:48:53 2014
new/usr/src/lib/libm/common/C/jn.c
libm - more cstyle fixes
common/C/jn.c: cstyle fixes
patch07 - removed dead code with mtsk.h
patch06 - libm: fixed compilation issues after updates
patch01 - 693 import Sun Devpro Math Library
libm - more cstyle fixes
common/C/jn.c: cstyle fixes
patch07 - removed dead code with mtsk.h
patch06 - libm: fixed compilation issues after updates
patch01 - 693 import Sun Devpro Math Library
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" 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 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 = 5.64189583547756286948079451560772585844l-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;
        }
        if (isnan(x))
                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));
        }
        if (n == 0)
                return (j0(x));
        if (n == 1)
                return (j1(x));
        if ((n&1) == 0)
                sgn = 0;                        /* even n */
        else
                sgn = signbit(x);       /* old n  */
        x = fabs(x);
        if (x == zero||!finite(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 > 1.0e91) {
                                /*
                                 * 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
```

```
119                                 *         1    -s-c            -c+s
120                                 *         2    -s+c            -c-s
121                                 *         3     s+c             c-s
122                                 */
123                 switch (n&3) {
124                     case 0: temp =  cos(x)+sin(x); break;
125                     case 1: temp = -cos(x)+sin(x); break;
126                     case 2: temp = -cos(x)-sin(x); break;
127                     case 3: temp =  cos(x)-sin(x); break;
128                 }
129                 b = invsqrtpi*temp/sqrt(x);
130             } else {
131                         a = j0(x);
132                         b = j1(x);
133                         for (i = 1; i < n; i++) {
134                     temp = b;
135                     b = b*((GENERIC)(i+i)/x) - a; /* avoid underflow */
136                     a = temp;
137                         }
138             }
139         } else {
140             if (x < 1e-9) {     /* use J(n,x) = 1/n!*(x/2)^n */
141                 b = pow(0.5*x, (GENERIC) n);
142                 if (b != zero) {
143                     for (a = one, i = 1; i <= n; i++) a *= (GENERIC)i;
144                     b = b/a;
145                 }
146             } else {
147                 /*
148                  * use backward recurrence
149                  *                          x         x^2     x^2
150                  *  J(n,x)/J(n-1,x) =  ----   ------   ------   .....
151                  *                          2n  - 2(n+1) - 2(n+2)
152                  *
153                  *                          1         1         1
154                  *  (for large x)   =  ----  ------   ------   .....
155                  *                          2n   2(n+1)   2(n+2)
156                  *                          -- - ------ - ------ -
157                  *                           x       x         x
158                  *
159                  * Let w = 2n/x and h = 2/x, then the above quotient
160                  * is equal to the continued fraction:
161                  *                   1
162                  *      = -----------------------
163                  *                  1
164                  *         w - -----------------
165                  *                    1
166                  *                 w+h - ---------
167                  *                    w+2h - ...
168                  *
169                  * To determine how many terms needed, let
170                  * Q(0) = w, Q(1) = w(w+h) - 1,
171                  * Q(k) = (w+k*h)*Q(k-1) - Q(k-2),
172                  * When Q(k) > 1e4      good for single
173                  * When Q(k) > 1e9      good for double
174                  * When Q(k) > 1e17     good for quaduple
175                  */
176             /* determin k */
177                 GENERIC t, v;
178                 double q0, q1, h, tmp; int k, m;
179                 w  = (n+n)/(double)x; h = 2.0/(double)x;
180                 q0 = w;  z = w + h; q1 = w*z - 1.0; k = 1;
181                 while (q1 < 1.0e9) {
182                         k += 1; z += h;
183                         tmp = z*q1 - q0;
184                         q0 = q1;
```



```
185                         q1 = tmp;
186                 }
187                 m = n+n;
188                 for (t = zero, i = 2*(n+k); i >= m; i -= 2) t = one/(i/x-t);
189                 a = t;
190                 b = one;
191                 /*
192                  * estimate log((2/x)^n*n!) = n*log(2/x)+n*ln(n)
193                  *  hence, if n*(log(2n/x)) > ...
194                  *  single 8.8722839355e+01
195                  *  double 7.09782712893383973096e+02
196                  *  long double 1.135652340629414394949193107797076500617 0e+04
197                  *  then recurrent value may overflow and the result is
198                  *  likely underflow to zero
199                  */
200                 tmp = n;
201                 v = two/x;
202                 tmp = tmp*log(fabs(v*tmp));
203                 if (tmp < 7.09782712893383973096e+02) {
204                         for (i = n-1; i > 0; i--) {
205                             temp = b;
206                             b = ((i+i)/x)*b - a;
207                         a = temp;
208                             }
209                 } else {
210                             for (i = n-1; i > 0; i--) {
211                                 temp = b;
212                                 b = ((i+i)/x)*b - a;
213                                 a = temp;
214                                     if (b > 1e100) {
215                                             a /= b;
216                                             t /= b;
217                                             b  = 1.0;
218                                     }
219                             }
220                 }
221                         b = (t*j0(x)/b);
222             }
223         }
224         if (sgn == 1)
225                 return (-b);
226         else
227                 return (b);
228 }

230 GENERIC
231 yn(int n, GENERIC x) {
232         int i;
233         int sign;
234         GENERIC a, b, temp = 0, ox, on;

236         ox = x; on = (GENERIC)n;
237         if (isnan(x))
238                 return (x*x);   /* + -> * for Cheetah */
239         if (x <= zero) {
240                 if (x == zero) {
241                         /* return -one/zero; */
242                         return (_SVID_libm_err((GENERIC)n, x, 12));
243                 } else {
244                         /* return zero/zero; */
245                         return (_SVID_libm_err((GENERIC)n, x, 13));
246                 }
247         }
248         if (!((int) _lib_version == libm_ieee ||
249                 (__xpg6 & _C99SUSv3_math_errexcept) != 0)) {
250             if (x > X_TLOSS)
```

```
251                         return (_SVID_libm_err(on, ox, 39));
252         }
253         sign = 1;
254         if (n < 0) {
255                 n = -n;
256                 if ((n&1) == 1) sign = -1;
257         }
258         if (n == 0)
259                 return (y0(x));
260         if (n == 1)
261                 return (sign*y1(x));
262         if (!finite(x))
263                 return (zero);

265         if (x > 1.0e91) {
266                                 /*
267                                  * x >> n**2
268                                  *  Jn(x) = cos(x-(2n+1)*pi/4)*sqrt(2/x*pi)
269                                  *  Yn(x) = sin(x-(2n+1)*pi/4)*sqrt(2/x*pi)
270                                  *  Let s = sin(x), c = cos(x),
271                                  *  xn = x-(2n+1)*pi/4, sqt2 = sqrt(2), then
272                                  *
273                                  *    n sin(xn)*sqt2    cos(xn)*sqt2
274                                  *      ----------------------------------
275                                  *       0       s-c             c+s
276                                  *       1      -s-c            -c+s
277                                  *       2      -s+c            -c-s
278                                  *       3       s+c             c-s
279                                  */
280                 switch (n&3) {
281                     case 0: temp =  sin(x)-cos(x); break;
282                     case 1: temp = -sin(x)-cos(x); break;
283                     case 2: temp = -sin(x)+cos(x); break;
284                     case 3: temp =  sin(x)+cos(x); break;
285                 }
286                 b = invsqrtpi*temp/sqrt(x);
287         } else {
288                 a = y0(x);
289                 b = y1(x);
290                 /*
291                  * fix 1262058 and take care of non-default rounding
292                  */
293                 for (i = 1; i < n; i++) {
294                         temp = b;
295                         b *= (GENERIC) (i + i) / x;
296                         if (b <= -DBL_MAX)
297                                 break;
298                         b -= a;
299                         a = temp;
300                 }
301         }
302         if (sign > 0)
303                 return (b);
304         else
305                 return (-b);
306 }
```

```
**********************************************************
    1118 Thu Oct  9 19:48:53 2014
new/usr/src/lib/libm/common/C/matherr.c
libm - more cstyle fixes
patch01 - 693 import Sun Devpro Math Library
libm - more cstyle fixes
patch01 - 693 import Sun Devpro Math Library
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" 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 matherr = __matherr

#include "libm.h"

/* ARGSUSED0 */
int
__matherr(struct exception *x) {
        return (0);
}
```

```
**********************************************************
    2079 Thu Oct  9 19:48:53 2014
new/usr/src/lib/libm/common/C/sinh.c
libm - more cstyle fixes
patch01 - 693 import Sun Devpro Math Library
libm - more cstyle fixes
patch01 - 693 import Sun Devpro Math Library
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" 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.
 */

#pragma weak sinh = __sinh

/* INDENT OFF */
/*
 * sinh(x)
 * Code originated from 4.3bsd.
 * Modified by K.C. Ng for SUN 4.0 libm.
 * 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-1024*ln2)*2**1023
 *
 *
 * Special cases:
 *      sinh(x) is x if x is +INF, -INF, or NaN.
 *      only sinh(0)=0 is exact for finite argument.
 *
 */
/* INDENT ON */

#include "libm.h"

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



```
double
sinh(double x) {
        double  ox, r, t;

        ox = x;
        r = fabs(x);
        if (!finite(x))
                return (x * r);
        if (r < lnovft) {
                t = expm1(r);
                r = copysign((t + t / (1.0 + t)) * 0.5, x);
        } else {
                if (r < 1000.0)
                        x = copysign(exp((r - 1024 * ln2hi) - 1024 * ln2lo), x);
                r = scalbn(x, 1023);
        }
        if (!finite(r))
                r = _SVID_libm_err(ox, ox, 25);
        return (r);
}
```

```
**********************************************************
    2521 Thu Oct  9 19:48:53 2014
new/usr/src/lib/libm/common/C/tanh.c
libm - more cstyle fixes
patch01 - 693 import Sun Devpro Math Library
libm - more cstyle fixes
patch01 - 693 import Sun Devpro Math Library
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" 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 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_synonyms.h"
#include "libm_protos.h"
#include <math.h>

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;

        if (isnan(x))
                return (x * x); /* + -> * for Cheetah */
        signx = signbit(x);
        t = fabs(x);
        z = one;
        if (t <= 22.0) {
                if (t > one)
                        z = one - two / (expm1(t + t) + two);
                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))
                return (copysign(1.0, x));
        else
                return (signx == 1 ? -z + small * small : z - small * small);

        return (signx == 1 ? -z : z);
}
```

```
**********************************************************
   14555 Thu Oct  9 19:48:54 2014
new/usr/src/lib/libm/common/LD/__lgammal.c
libm - cstyle fixes
patch01 - 693 import Sun Devpro Math Library
libm - cstyle fixes
patch01 - 693 import Sun Devpro Math Library
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" 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.
 */

/*
 * 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 "libm_synonyms.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;

long double
__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);
        *signgamlp = 1;
        if (signbitl(x))
                return (neg(x, signgamlp));

        /* for x < 8.0 */
        if (x < 8.0L) {
            y = anintl(x);
            i = (int) y;
            switch (i) {
            case 0:
                if (x < 1.0e-40L)
                        return (-logl(x));
                else
                        return (primary(x)-log1pl(x))-logl(x);
            case 1:
                return (primary(x-y)-logl(x));
            case 2:
                return (primary(x-y));
            case 3:
                return (primary(x-y)+logl(x-c1));
            case 4:
                return (primary(x-y)+logl((x-c1)*(x-c2)));
            case 5:
                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)));
            case 7:
                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));
            }
        }

        /* 8.0 <= x < 1.0e40 */
        if (x < 1.0e40L) {
            t = logl(x);
            return (x*(t-c1)-(ch*t-polytail(c1/x)));
        }

        /* 1.0e40 <= x <= inf */
        return (x*(logl(x)-c1));
}

static const long double an1[] = {              /* 20 terms */
```

```
        -0.07721566490153286060651209008240243097 41L,
        3.22467033424113218236207583323013028 9059e-0001L,
        -6.73523010531980951332460538366892996 4120e-0002L,
        2.05808084277845478790009243292891022 6297e-0002L,
        -7.38555102867398526627305408608110212 5704e-0003L,
        2.89051033074152328575886730440962864 8727e-0003L,
        -1.19275391170326097658141433809626749 8555e-0003L,
        5.09669524743042456283195666285569782 4035e-0004L,
        -2.23154758453577797892679850208430012 3638e-0004L,
        9.94575127818638467027826803432215794 7635e-0005L,
        -4.49262367366554772664783847412514763 1082e-0005L,
        2.05072128061779681009699315428156116 8706e-0005L,
        -9.43948778561739655209239323404476731 3568e-0006L,
        4.37487290351605151068923417313979315 9340e-0006L,
        -2.03915667641364309104045982577602932 7487e-0006L,
        9.55577718131862147046656354380621152 3634e-0007L,
        -4.46834491970963063755853831348239898 9638e-0007L,
        2.21673808609004578177300447783105944 4178e-0007L,
        -7.47278340341838845586044584254384348 5916e-0008L,
        8.77731793092714992205678213270623892 1648e-0008L,
};

static const long double an2[] = {              /* 20 terms */
  -.07721566490153286060626927236981276070 18L,
   3.22467033424113218263555234906027911804 7e-0001L,
  -6.73523010531980936755564288313399481832 5e-0002L,
   2.05808084277845967688082220276214367181 3e-0002L,
  -7.38555102867282821601134315007784691893 0e-0003L,
   2.89051033076206060739956153690572785317 8e-0003L,
  -1.19275391141962326232818753275975636804 1e-0003L,
   5.09669527863645667825809113453225861861 4e-0004L,
  -2.23154730681753574305297519402289336913 5e-0004L,
   9.94577146163331328274426485398664387708 7e-0005L,
  -4.49250327945897203792687606125748948161 9e-0005L,
   2.05131141681208287549267865136939459561 3e-0005L,
  -9.41577828236595520391585076153746294116 5e-0006L,
   4.45242882904514709872293298108865005591 9e-0006L,
  -1.83502472798763257988695176065072269578 1e-0006L,
   1.37978308065854500957906071494638146256 5e-0006L,
   2.28263753210977515676973676874840217523 8e-0007L,
   1.00257737551590019136211971812814988016 8e-0006L,
   5.17702879426263831193999110642322000246 3e-0007L,
   3.12794724517484710412242644593783055575 5e-0007L,
};

static const long double an3[] = {              /* 20 terms */
  -.07721566490153282278706464177292206908 75L,
   3.22467033424115669988178895595991525036 5e-0001L,
  -6.73523010531227357137543105974497556317 0e-0002L,
   2.05808084292446458766284607133708380900 5e-0002L,
  -7.38555100867727165472360465395613179161 9e-0003L,
   2.89051053647978208619711027258383317660 2e-0003L,
  -1.19275226207685769274057156780825913869 7e-0003L,
   5.09680077114980528937113515512838070788 9e-0004L,
  -2.23100083668283133550505849240986012364 7e-0004L,
   9.96891217107393680387180396636059527504 7e-0005L,
  -4.41202077932774624354438794616725618725 8e-0005L,
   2.28137411354145415106701663299863020904 9e-0005L,
  -4.02836129142862949182469655287954266830 e-0006L,
   1.47069492061951892459895684922653075013 9e-0005L,
   1.38168613761798719795289545582377713772 e-0005L,
   2.01249353926577772894475998205497044160 1e-0005L,
   1.72391786420896549025156064468193367579 9e-0005L,
   1.20295403524378830013860876542512371339 5e-0005L,
   5.07985188755862309277629657703085093814 6e-0006L,
   1.22065794582415375155513859200660402628 2e-0006L,
```



```
};

static const long double an4[] = {              /* 21 terms */
  -.07721566490157322853502618166975403923 71L,
   3.22467033422175206069175134036521222609 7e-0001L,
  -6.73523010974400969397775599148819636827 9e-0002L,
   2.05808077891303762690995414161158078321 6e-0002L,
  -7.38555756793150562117048370895055750681 9e-0003L,
   2.89045983841625432634084428978525488343 6e-0003L,
  -1.19305903620713676287735159696671845573 7e-0003L,
   5.08191470810037283661337135652956893786 9e-0004L,
  -2.28985501613360031313155300598254204533 8e-0004L,
   8.05345453798058587962033105383349851149 1e-0005L,
  -9.57462053210484582124349340585567243899 8e-0005L,
  -9.26908562820710715560144500119631771568 6e-0005L,
  -2.18327677985949046171619634477620822018 0e-0004L,
  -3.13483430559757109645245499973726966886 8e-0004L,
  -3.97387889495193743701830598690139288861 9e-0004L,
  -3.95335241489922279916127556438648805711 9e-0004L,
  -3.13674093220403877936266090062121281651 1e-0004L,
  -1.88450225381963407394613082519607862766 4e-0004L,
  -2.18327677985949046171619634477620822018 0e-0004L,
  -1.19265579995892685358533254212363137930 1e-0005L,
  -2.29218375001057106289160507428174485443 6e-0005L,
  -3.22398062872971686492772426578140661429 4e-0006L,
};

static const long double ap1[] = {                      /* 19 terms */
  -0.07721566490153286060651209008240242961L,
   3.22467033424113218236207583323004795646 5e-0001L,
  -6.73523010531980951332460538296394377730 1e-0002L,
   2.05808084277845478790009212660625237546 5e-0002L,
  -7.38555102867398526627251823136502006394 1e-0003L,
   2.89051033074152328568170457079770736423 e-0003L,
  -1.19275391170326097128530422116599024451 5e-0003L,
   5.09669524743042087869601818883088697224 5e-0004L,
  -2.23154758453565400464763973784152602509 5e-0004L,
   9.94575127813720196063609880585231598291 9e-0005L,
  -4.49262367277760605358791946392904422628 0e-0005L,
   2.05072125870328948760370267075305376520 1e-0005L,
  -9.43948562656561698935275067249900802104 1e-0006L,
   4.37483816240399464513820041935684457421 9e-0006L,
  -2.03897949286255534857700694445100216149 6e-0006L,
   9.53676315238226354808698119137888510280 2e-0007L,
  -4.42611121433243404986359523191656401491 3e-0007L,
   1.91114884751294746423463384627028754688 2e-0007L,
  -5.78867394486192303815783908027230351967 1e-0008L,
};

static const long double ap2[] = {                      /* 19 terms */
  -0.07721566490153286060642862444935483608 7L,
   3.22467033424113218227194874426585544013 9e-0001L,
  -6.73523010531980946735612659900505167620 3e-0002L,
   2.05808084277845331571638981521349600258 8e-0002L,
  -7.38555102867365332306411842258009622295 9e-0003L,
   2.89051033073592357208800342484928900603 9e-0003L,
  -1.19275391162995236860618554394579068814 4e-0003L,
   5.09669523980671887536454758704322099876 6e-0004L,
  -2.23154752060061610899186712739208914488 6e-0004L,
   9.94574691389815112061232283059416008973 e-0005L,
  -4.49259930746197700357022494305458572968 4e-0005L,
   2.05060989188916545359204650565175999090 e-0005L,
  -9.43532986673419379654051524791716598857 9e-0006L,
   4.36226713852222323624101613658565144581 e-0006L,
  -2.00855635665324657930049160149751023055 7e-0006L,
   8.96149810338720716110534711804284435 4395e-0007L,
  -3.61418722833021628223569280648834115774 1e-0007L,
```

```
257    1.13697898824781686050042091501477753153e-0007L,
258   -2.00053278638719666401928651489978269177 6e-0008L,
259 };

261 static const long double ap3[] = {                     /* 19 terms */
262   -0.07721566490153285988852147079534885644 6L,
263    3.22467033424113173336404861448422844307 7e-0001L,
264   -6.73523010531967654166049514525903815157 6e-0002L,
265    2.05808084277597546183776883901544427383 0e-0002L,
266   -7.38555102834761572972861806666356606906e-0003L,
267    2.89051032751795408337903200864308025667 6e-0003L,
268   -1.19275388691947072800182113743943088260 3e-0003L,
269    5.09669372889893223481490376914657748291 2e-0004L,
270   -2.23154005504882766252859401096187425803 7e-0004L,
271    9.94544621001864931149161999943883384372 3e-0005L,
272   -4.49160820659806451919023624575386769775 0e-0005L,
273    2.04793907132227101649806505285374646666 9e-0005L,
274   -9.37682404652278600667754103663153679076 2e-0006L,
275    4.25932982949814911158227720918915012734 7e-0006L,
276   -1.86606477042159426670217628976421287342 8e-0006L,
277    7.46206672113757959292812810453495713566 9e-0007L,
278   -2.48354621752907773507400713845767872737 1e-0007L,
279    5.91516657637816147329932467364914429757 4e-0008L,
280   -7.33413964170698896696625233375960470190 5e-0009L,
281 };

283 static const long double ap4[] = {                     /* 19 terms */
284   -0.07721566490153267855693132526372386736 75L,
285    3.22467033424105143500884268572246834482 2e-0001L,
286   -6.73523010530283200747943177216094849925 4e-0002L,
287    2.05808084255348118364852936096744188991 2e-0002L,
288   -7.38555100760290924202470680465987919924 4e-0003L,
289    2.89051018247390725393982131224830347120 6e-0003L,
290   -1.19275309842785677084789449758682561445 0e-0003L,
291    5.09665963641881156806333921420369355080 4e-0004L,
292   -2.23142114400435569116619425967500448363 9e-0004L,
293    9.94207384234383213275433288183387625136e-0005L,
294   -4.48380926197320453126325655050701205397e-0005L,
295    2.03326014261028488831911665493199444717 3e-0005L,
296   -9.15353954402664669987052819141044058579 6e-0006L,
297    3.98846046992548272589414468869958499797 1e-0006L,
298   -1.60969298008702917256795722185082597762 1e-0006L,
299    5.63491637724997582539970669449668880348 8e-0007L,
300   -1.56006546592951856354908320848259143769 6e-0007L,
301    2.96135019386893532552696220901938782158 4e-0008L,
302   -2.83460221519536813010464923450503315984 2e-0009L,
303 };

305 static long double
306 primary(long double s) {        /* assume |s|<=0.5 */
307         int i;

309         i = (int) (8.0L * (s + 0.5L));
310         switch (i) {
311         case 0: return ch*s+s*poly(s, an4, 21);
312         case 1: return ch*s+s*poly(s, an3, 20);
313         case 2: return ch*s+s*poly(s, an2, 20);
314         case 3: return ch*s+s*poly(s, an1, 20);
315         case 4: return ch*s+s*poly(s, ap1, 19);
316         case 5: return ch*s+s*poly(s, ap2, 19);
317         case 6: return ch*s+s*poly(s, ap3, 19);
318         case 7: return ch*s+s*poly(s, ap4, 19);
319         }
320         /* NOTREACHED */
321     return (0.0L);
322 }
```



```
324 static long double
325 poly(long double s, const long double *p, int n) {
326         long double y;
327         int i;
328         y = p[n-1];
329         for (i = n-2; i >= 0; i--) y = p[i]+s*y;
330         return (y);
331 }

333 static const long double pt[] = {
334    9.18938533204672741780329736405617680466 3e-0001L,
335    8.33333333333333333333333333331286969123e-0002L,
336   -2.77777777777777777777777755319479603640 2e-0003L,
337    7.93650793650793650792728307143358424817 6e-0004L,
338   -5.95238095238095236235104216319263410829 7e-0004L,
339    8.41750841750839566174286645578379460131e-0004L,
340   -1.91752691752526365118606641793468567564 9e-0003L,
341    6.41025640939520316465929297314229319908 3e-0003L,
342   -2.95506532724830330176359451401241843818 8e-0002L,
343    1.79644283009906754294599861541189382288 6e-0001L,
344   -1.39241346582972374248997431041111866291 9e+0000L,
345    1.33998423803726765835265659796049202926 1e+0001L,
346   -1.56470765760537366242578590427864572781 3e+0002L,
347    2.15632380749921135612781396222306707930 0e+0003L,
348   -3.33048642762622318464729983413704130756 9e+0004L,
349    5.23553507201188921361136925414012351869 9e+0005L,
350   -7.25816098460222071049198857343021259308 0e+0006L,
351    7.31652693456968645964143888234032267335 7e+0007L,
352   -3.80645027906490054883657178928489671147 3e+0008L,
353 };

355 static long double
356 polytail(long double s) {
357         long double t, z;
358         int i;
359         z = s*s;
360         t = pt[18];
361         for (i = 17; i >= 1; i--) t = pt[i]+z*t;
362         return (pt[0]+s*t);
363 }

365 static long double
366 neg(long double z, int *signgamlp) {
367         long double t, p;

369         /*
370          * written by K.C. Ng,  Feb 2, 1989.
371          *
372          * Since
373          *                  -z*G(-z)*G(z) = pi/sin(pi*z),
374          * we have
375          *      G(-z) = -pi/(sin(pi*z)*G(z)*z)
376          *                =  pi/(sin(pi*(-z))*G(z)*z)
377          * Algorithm
378          *              z = |z|
379          *              t = sinpi(z); ...note that when z>2**112, z is an int
380          *              and hence t=0.
381          *
382          *              if(t==0.0) return 1.0/0.0;
383          *              if(t< 0.0) *signgamlp = -1; else t= -t;
384          *              if(z<1.0e-40)   ...tiny z
385          *                  return -log(z);
386          *              else
387          *                  return log(pi/(t*z))-lgamma(z);
388          *
```

```
389          */
391         t = sinpil(z);                  /* t := sin(pi*z) */
392         if (t == c0)                    /* return   1.0/0.0 =  +INF */
393             return (c1/c0);
395         z = -z;
396         if (z <= tiny)
397             p = -logl(z);
398         else
399                 p = logl(pi/(fabsl(t)*z)) - __k_lgammal(z, signgamlp);
400         if (t < c0) *signgamlp = -1;
401         return (p);
402 }
```

```
**********************************************************
    1637 Thu Oct  9 19:48:54 2014
new/usr/src/lib/libm/common/LD/asinhl.c
fix lint warnings in LD/asinhl.c and LD/tanhl.c
asinhl.c
patch07 - removed dead code with mtsk.h
patch06 - libm: fixed compilation issues after updates
patch01 - 693 import Sun Devpro Math Library
fix lint warnings in LD/asinhl.c and LD/tanhl.c
asinhl.c
patch07 - removed dead code with mtsk.h
patch06 - libm: fixed compilation issues after updates
patch01 - 693 import Sun Devpro Math Library
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" 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.
 */

#if defined(ELFOBJ)
#pragma weak asinhl = __asinhl
#endif

#include "libm.h"

static const long double
        ln2     = 6.93147180559945309417232121458176568075e-0001L,
        one     = 1.0L,
        big     = 1.0e+20L,
        tiny    = 1.0e-20L;

long double
asinhl(long double x) {
        long double t, w;
#ifndef lint
        volatile long double dummy;
#endif

        w = fabsl(x);
        if (isnanl(x))
                return (x + x); /* x is NaN */
        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
                return (copysignl(logl(w) + ln2, x));
}
```

```
**********************************************************
   27946 Thu Oct  9 19:48:54 2014
new/usr/src/lib/libm/common/LD/j0l.c
libm - cstyle fixes
minor changes
patch07 - removed dead code with mtsk.h
patch06 - libm: fixed compilation issues after updates
libm fixes from richlowe - richlowe.net/webrevs/il_keith
patch01 - 693 import Sun Devpro Math Library
libm - cstyle fixes
minor changes
patch07 - removed dead code with mtsk.h
patch06 - libm: fixed compilation issues after updates
libm fixes from richlowe - richlowe.net/webrevs/il_keith
patch01 - 693 import Sun Devpro Math Library
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" 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.
 */

/*
 * 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 "libm_synonyms.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,
eight   = 8.0L,
invsqrtpi = 5.64189583547756286948079451560772585844 1e-0001L,
tpi     = 0.63661977236758134307553505349005744 8L;

static GENERIC pzero(), qzero();
static GENERIC r0[7] = {
  -2.49999999999999999999999999998934492e-0001L,
   1.27265792736004978632761845113376371488 0e-0002L,
  -2.69449976371296327690063669340065960089 8e-0004L,
   2.72487747505897757690323407091961644788 3e-0006L,
  -1.43261710321433023696747749539307632028 1e-0008L,
   3.82324880408007916870668354051379222447 1e-0011L,
  -4.18317427756798364733756850428631366506 5e-0014L,
};
static GENERIC s0[7] = {
   1.0e0L,
   1.15936829055980085468952619546288466639 5e-0002L,
   6.62939759739497338300974387616994677255 9e-0005L,
   2.42677998139405440630543114250173509434 0e-0007L,
   6.09766349124851106909440046963544974988 3e-0010L,
   1.01701913334092922023874741321605222403 6e-0012L,
   9.01259317930619757951837458196937127848 1e-0016L,
};

GENERIC
j0l(x) GENERIC x; {
        GENERIC z, s, c, ss, cc, r, u, v;
        int i;

        if (isnanl(x))
                return (x+x);
        x = fabsl(x);
        if (x > 1.28L) {
                if (!finitel(x))
                        return (zero);
                s = sinl(x);
                c = cosl(x);
        /*
         * 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.
         */
                if (x > 1.0e2450L) {    /* x+x may overflow */
                        ss = s-c;
                        cc = s+c;
                } else if (signbitl(s) != signbitl(c)) {
                        ss = s - c;
                        cc = -cosl(x+x)/ss;
                } else {
                        cc = s + c;
                        ss = -cosl(x+x)/cc;
```

```
117                 }
118         /*
119          * j0(x) = 1/sqrt(pi) * (P(0,x)*cc - Q(0,x)*ss) / sqrt(x)
120          * y0(x) = 1/sqrt(pi) * (P(0,x)*ss + Q(0,x)*cc) / sqrt(x)
121          */
122                 if (x > 1.0e120L)
123                         return (invsqrtpi*cc)/sqrtl(x);
124                 u = pzero(x); v = qzero(x);
125                 return (invsqrtpi*(u*cc-v*ss)/sqrtl(x));
126         }
127         if (x <= small) {
128             if (x <= tiny)
129                         return (one-x);
130             else
131                         return (one-x*x*0.25L);
132         }
133         z = x*x;
134         r = r0[6]; s = s0[6];
135         for (i = 5; i >= 0; i--) {
136             r = r*z + r0[i];
137             s = s*z + s0[i];
138         }
139         return (one+z*(r/s));
140 }

141 static GENERIC u0[8] = {
142   -7.38042951086872252743439279484830163122 0e-0002L,
143    1.76685555962594079185753694930198181651 3e-0001L,
144   -1.38647072270104792323555325124016283940 8e-0002L,
145    3.52014924272481157863697081163122486261 5e-0004L,
146   -3.97859966324379004985364227562495187002 5e-0006L,
147    2.22880115263957224547222556806915479763 e-0008L,
148   -6.12124676429878501865859717949883373161 77e-0011L,
149    6.67710362972267883347596581052558739659 6e-0014L,
150 };
151 static GENERIC v0[8] = {
152    1.0e0L,
153    1.24716441653911131157167676612776712797 0e-0002L,
154    7.82914474963979150005290028148936744357 6e-0005L,
155    3.24712654042224533051121832101336033660 6e-0007L,
156    9.75051672478949967856706257254956844786 9e-0010L,
157    2.15671322317359121225054339025845809877 6e-0012L,
158    3.32216956159789000423148243123645275262 4e-0015L,
159    2.82121329531400092425222648630572680509 3e-0018L,
160 };

161 GENERIC
162 y0l(x) GENERIC x; {
163         GENERIC z, d, s, c, ss, cc, u, v;
164         int i;

165         if (isnanl(x))
166                 return (x+x);
167         if (x <= zero) {
168                 if (x == zero)
169                     d = -one/(x-x);
170                 else
171                     d = zero/(x-x);
172         }
173         if (x > 1.28L) {
174                 if (!finitel(x))
175                         return (zero);
176                 s = sinl(x);
177                 c = cosl(x);
178         /*
179          * j0(x) = sqrt(2/(pi*x))*(p0(x)*cos(x0)-q0(x)*sin(x0))
```



```
180          * where x0 = x-pi/4
181          *      Better formula:
182          *              cos(x0) = cos(x)cos(pi/4)+sin(x)sin(pi/4)
183          *                      = 1/sqrt(2) * (cos(x) + sin(x))
184          *              sin(x0) = sin(x)cos(pi/4)-cos(x)sin(pi/4)
185          *                      = 1/sqrt(2) * (sin(x) - cos(x))
186          * To avoid cancellation, use
187          *              sin(x) +- cos(x) = -cos(2x)/(sin(x) -+ cos(x))
188          * to compute the worse one.
189          */
190                 if (x > 1.0e2450L) {    /* x+x may overflow */
191                         ss = s-c;
192                         cc = s+c;
193                 } else if (signbitl(s) != signbitl(c)) {
194                         ss = s - c;
195                         cc = -cosl(x+x)/ss;
196                 } else {
197                         cc = s + c;
198                         ss = -cosl(x+x)/cc;
199                 }
200         /*
201          * j0(x) = 1/sqrt(pi*x) * (P(0,x)*cc - Q(0,x)*ss)
202          * y0(x) = 1/sqrt(pi*x) * (P(0,x)*ss + Q(0,x)*cc)
203          */
204                 if (x > 1.0e120L)
205                         return (invsqrtpi*ss)/sqrtl(x);
206                 return (invsqrtpi*(pzero(x)*ss+qzero(x)*cc)/sqrtl(x));

207         }
208         if (x <= tiny) {
209             return (u0[0] + tpi*logl(x));
210         }
211         z = x*x;
212         u = u0[7]; v = v0[7];
213         for (i = 6; i >= 0; i--) {
214             u = u*z + u0[i];
215             v = v*z + v0[i];
216         }
217         return (u/v + tpi*(j0l(x)*logl(x)));
218 }

219 static GENERIC pr0[12] = {      /* [16 -- inf] */
220    9.99999999999999999999999999999997515e-0001L,
221    1.06598161537727337642536582396755059835 8e-0003L,
222    4.39099120092758897830637471898424071913 0e-0005L,
223    9.07208621860798671184706940733932136310 3e-0007L,
224    1.02255288617737536740840850104646167152 8e-0010L,
225    6.42076691224365824157063585408959726903 1e-0011L,
226    2.20645172512693391359108021108124226690 8e-0013L,
227    3.92836959681689507736370547874334629836 8e-0014L,
228    3.25815992887412459728670111972148287659 6e-0015L,
229    1.02571580813418897886067913014068510134 8e-0016L,
230    7.53717087479572125579600168702403128068 5e-0015L,
231   -1.57941390145015733230774558600420768779 6e-0014L,
232 };
233 static GENERIC ps0[11] = {
234    1.0e0L,
235    1.06605192787727337642536582396755051268 7e-0003L,
236    4.39173964716838159239917380432926635303 8e-0005L,
237    9.07516226180134367180565829412388886788 4e-0007L,
238    1.02318611851990475181958191207598599505 8e-0010L,
239    6.42786186041422374634051537651273027506 1e-0011L,
240    2.21086150323782358973548130362799340623 5e-0013L,
241    3.94324733578429290591595684090181817798 9e-0014L,
242    3.28372097677754514215020011064727000448 1e-0015L,
243    1.04534691881275404890364564153872898675 9e-0016L,
```

```
249     8.043455468065618900750599584291193680463e+0015L,
250 };
251 static GENERIC pr1[12] = {      /* [8 -- 16] */
252     9.99999999999999999997844227011086836l8e-0001L,
253     6.79609853294833420775548869277790706289 4e+0002L,
254     1.84003611260572216882453075879716983604 2e+0005L,
255     2.59849048319191663726489434063584759812 2e+0007L,
256     2.10577486324270702552573024947205457852 3e+0009L,
257     1.01582204423054242666631499779694497959 e+0011L,
258     2.93155745700811043676407769994418907187 5e+0012L,
259     4.96288512112545763365525922417932280882 4e+0013L,
260     4.70542405514822326915543059856335156627 9e+0014L,
261     2.29443985491074722915205608091042700111 0e+0015L,
262     4.90553184313748669150095001932247545862 9e+0015L,
263     3.18754316971033921879344254284573599456 5e+0015L,
264 };
265 static GENERIC ps1[14] = {
266     1.0e0L,
267     6.79680165794833420775457157606675818028 8e+0002L,
268     1.84051289120130056732542105982667636644 7e+0005L,
269     2.59977702831291897530625216712769507522 1e+0007L,
270     2.10758257277104763684681128463424489253 7e+0009L,
271     1.01727579469415610897578276388979940348 e+0011L,
272     2.93848764519246384542805975545476231601 1e+0012L,
273     4.98251216473555705452104291618231792446 6e+0013L,
274     4.73763990015370327479267746826456436143 7e+0014L,
275     2.32339871912374274352424952827509710064 6e+0015L,
276     5.03341910706921057786890979789698441939 1e+0015L,
277     3.40903610593106860960131707675980471605 9e+0015L,
278     7.50565536435267973758574514775352166216 6e+0013L,
279   -9.97683715398368825078019824829710911831 3e+0012L,
280 };
281 static GENERIC pr2[12] = {      /* [5 -- 8 ] */
282     9.99999999999999993785723678927736632022 0e-0001L,
283     3.69284876526864957165160242037635884921 4e+0002L,
284     5.37302206753547657692671590005776098541 0e+0004L,
285     4.03873889119131496997150403505721943072 5e+0006L,
286     1.72828570630694052339738556665976264699 9e+0008L,
287     4.37540081964588991115868873720605478853 4e+0009L,
288     6.59895041820491240837559121778208856707 6e+0010L,
289     5.82718203918323849248027540152007279378 3e+0011L,
290     2.88422264291349239088757241499949097584 4e+0012L,
291     7.37327887379776772193283783062868863277 5e+0012L,
292     8.33829545756897376120507796439796923048 9e+0012L,
293     2.91138318346728834577230881720980692214 3e+0012L,
294 };
295 static GENERIC ps2[14] = {
296     1.0e0L,
297     3.69355189026864947728889626717199321310 2e+0002L,
298     5.37560788099836150247471513382806851429 7e+0004L,
299     4.04247776402410824974499886257278636732 8e+0006L,
300     1.73106983873701695668583958867013293951 3e+0008L,
301     4.38714767404989877873822658593549141772 8e+0009L,
302     6.62805865962065376534955694056771525816 5e+0010L,
303     5.86965990416417774047168585636732216066 4e+0011L,
304     2.91983944622817017058977559638969436383 e+0012L,
305     7.53531489769667140262820371861230925390 7e+0012L,
306     8.69635556145293377577330985974861065893 5e+0012L,
307     3.21615510314153722117360155769708321625 7e+0012L,
308     4.75685708106894224824688015921378908636 3e+0010L,
309   -3.49635661966660803223107486648147282406 7e+0009L,
310 };
311 static GENERIC pr3[13] = {      /* [3.5 -- 5 ] */
312     9.99999999999991669310728561239819658824 7e-0001L,
313     2.26397592128291772119442532048497433694 5e+0002L,
314     1.99435838674424584888949276278148419996 6e+0004L,
```



```
315     8.98006745843054224355996249383166132316 8e+0005L,
316     2.28221378752137266370556775642008755350 8e+0007L,
317     3.40978437488906361825028866990837513592 3e+0008L,
318     3.02438085740144858925434351758981171110 8e+0009L,
319     1.57111036804674024689507172144308228637 9e+0010L,
320     4.60318702024360463215368530046316059376 8e+0010L,
321     7.08719645340971271944954928066405879340 3e+0010L,
322     5.04619602177634635680368740964423906504 1e+0010L,
323     1.28775843908016576570915427661885479993 2e+0010L,
324     5.90067977341502343378784665809681359078 4e+0008L,
325 };
326 static GENERIC ps3[13] = {
327     1.0e0L,
328     2.26467904628285506132860461923177474711 6e+0002L,
329     1.99593952398894455375565325538981210344 8e+0004L,
330     8.99385314470634872703838996749018323682 0e+0005L,
331     2.28832609963458884390698998370479546877 3e+0007L,
332     3.42496710025524088516924095680479011828 2e+0008L,
333     3.04631179797246399136802375964002891001 6e+0009L,
334     1.58961496193282681279022247970079722400 3e+0010L,
335     4.69240662452774481649708913932507393992 7e+0010L,
336     7.32048649590200891286646284907310832394 8e+0010L,
337     5.34594597282897828993530959774298136099 4e+0010L,
338     1.44403309191042375412130991509224717100 8e+0010L,
339     7.98771468511531466837895727382438361052 5e+0008L,
340 };
341 static GENERIC pr4[13] = {      /* [2.5 , 3.5] */
342     9.99999999998673667796111872274775712260e-0001L,
343     1.45382498070380055903787312356837884566 3e+0002L,
344     8.09732721643068228826761044700650866103 2e+0003L,
345     2.27384725203826437023116968638019266213 5e+0005L,
346     3.56105672804621111135475999897698544962 2e+0006L,
347     3.24493358880009637843462702936968037859 9e+0007L,
348     1.74011239286071795037621003890847679258 8e+0008L,
349     5.42617018745589328519787856388157926952 4e+0008L,
350     9.49010748645436232100437733602052628137 1e+0008L,
351     8.68887243942847004980171412107000531380 6e+0008L,
352     3.67331585316643722281191065690012321551 5e+0008L,
353     5.57777047035930330516487744633969327023 9e+0007L,
354     1.54043864203168964130819788018129186571 4e+0006L,
355 };
356 static GENERIC ps4[13] = {      /* [2.5 , 3.5] */
357     1.0e0L,
358     1.45452810569815943977303595195913179981 6e+0002L,
359     8.10744221520039239717217990043498785961 8e+0003L,
360     2.27939039377824288757417709660632899414 0e+0005L,
361     3.57625162559225200842478111177093413584 4e+0006L,
362     3.26790949905693263140594205867093381386 3e+0007L,
363     1.76002151533080553749977823809970464880 5e+0008L,
364     5.52555378766735398124206022258746572672 9e+0008L,
365     9.76987029591282045788938408267126932851 1e+0008L,
366     9.11058207100477427922690562962401800845 4e+0008L,
367     3.98185767862195559937196768034391845434 5e+0008L,
368     6.48240468623076939907319296166769703670 6e+0007L,
369     2.21004694309587840244353546032939178229 8e+0006L,
370 };
371 static GENERIC pr5[13] = {      /* [1.777..., 2.5] */
372     9.99999999911498610795181787114465588069 9e-0001L,
373     9.25258373604858834256834457031543594761 4e+0001L,
374     3.21872675785607871521463150240738626463 7e+0003L,
375     5.55400996462111165647958850586257704083 1e+0004L,
376     5.26999311564366433825319694452351029017 5e+0005L,
377     2.87461377377843069119291219061822054457 5e+0006L,
378     9.13353815110365835387414691961344243603 5e+0006L,
379     1.67306704141033892282519301307735424919 3e+0007L,
380     1.70691387384839801174479028920015184049 8e+0007L,
```

```
381    9.06776658385328853455160023557674761867 9e+0006L,
382    2.21674673345788456853269535503633865587 2e+0006L,
383    1.94575388080287254123570381272234451440 5e+0005L,
384    3.13237441292194807153919563888530951749 e+0003L,
385 };
386 static GENERIC ps5[13] = {      /* [1.777..., 2.5] */
387    1.0e0L,
388    9.25961498386218111888383167099034005298 2e+0001L,
389    3.22512527546290338484212407513260929030 4e+0003L,
390    5.57570536282901015452927600559418552464 92e+0004L,
391    5.30604986303708785549617012195844849252 2e+0005L,
392    2.90706075887350956430972990310901859721 5e+0006L,
393    9.29805920658499589829825782713120853928 9e+0006L,
394    1.72039107100696317683610802655654706298 0e+0007L,
395    1.78261481292286519047939450948794192061 2e+0007L,
396    9.70801638960527315353645203283987995015 5e+0006L,
397    2.47649508468817009648021564096217514002 7e+0006L,
398    2.36320066036558575966807779019460491718 7e+0005L,
399    4.80323956984819607712120357570435693673 1e+0003L,
400 };
401 static GENERIC pr6[13] = {      /* [1.28, 1.777...] */
402    9.99999996977709549599860692552432255955 6e-0001L,
403    5.82548671946619443050328382409687221921 6e+0001L,
404    1.24815549163775728191518482496537990538 0e+0003L,
405    1.30209319984235860932133841707171047761 5e+0004L,
406    7.35383580418629278284096199981054301603 9e+0004L,
407    2.35647166111368618054919509255575134175 7e+0005L,
408    4.35055326742900958163298706094278084710 1e+0005L,
409    4.58876266187660063871915982665238941823 5e+0005L,
410    2.67579639854852343654422104522529012861 1e+0005L,
411    8.07764955710897138829829291998844994046 4e+0004L,
412    1.11764045922130687351906874166405457377 6e+0004L,
413    5.54440007239681469517578751155775788558 5e+0002L,
414    5.07255054119148049843128908990582291071 8e+0000L,
415 };
416 static GENERIC ps6[13] = {      /* [1.28, 1.777...] */
417    1.0e0L,
418    5.83251792535716505063907584818361306329 1e+0001L,
419    1.25214436474359212817125610436497646689 8e+0003L,
420    1.31030023434221681357911802241558574077 2e+0004L,
421    7.43466769709381219781729215403286363292 3e+0004L,
422    2.39870659558771916572646900240400461471 1e+0005L,
423    4.47273751762510315700486937242748060251 1e+0005L,
424    4.78631352333776197529417142906703772361 1e+0005L,
425    2.85116187287273122847253606186536537019 2e+0005L,
426    8.89164826989914841233191802180138581558 6e+0004L,
427    1.29709748953535151757297812358475104228 7e+0004L,
428    7.09676164054597575620218414340046981261 8e+0002L,
429    8.37804933859023332597770240173334082035 1e+0000L,
430 };
431 static GENERIC sixteen = 16.0L;
432 static GENERIC huge    = 1.0e30L;

433 static GENERIC pzero(x)
434 GENERIC x;
435 {
436         GENERIC s, r, t, z;
437         int i;
438         if (x > huge)
439                 return (one);
440         t = one/x; z = t*t;
441         if (x > sixteen) {
442             r = z*pr0[11]+pr0[10]; s = ps0[10];
443             for (i = 9; i >= 0; i--) {
444                 r = z*r + pr0[i];
445                 s = z*s + ps0[i];
446             }
```



```
447         } else if (x > eight) {
448             r = pr1[11]; s = ps1[11]+z*(ps1[12]+z*ps1[13]);
449             for (i = 10; i >= 0; i--) {
450                 r = z*r + pr1[i];
451                 s = z*s + ps1[i];
452             }
453         } else if (x > five) {  /* x > 5.0 */
454             r = pr2[11]; s = ps2[11]+z*(ps2[12]+z*ps2[13]);
455             for (i = 10; i >= 0; i--) {
456                 r = z*r + pr2[i];
457                 s = z*s + ps2[i];
458             }
459         } else if (x > 3.5L) {
460             r = pr3[12]; s = ps3[12];
461             for (i = 11; i >= 0; i--) {
462                 r = z*r + pr3[i];
463                 s = z*s + ps3[i];
464             }
465         } else if (x > 2.5L) {
466             r = pr4[12]; s = ps4[12];
467             for (i = 11; i >= 0; i--) {
468                 r = z*r + pr4[i];
469                 s = z*s + ps4[i];
470             }
471         } else if (x > (1.0L/0.5625L)) {
472             r = pr5[12]; s = ps5[12];
473             for (i = 11; i >= 0; i--) {
474                 r = z*r + pr5[i];
475                 s = z*s + ps5[i];
476             }
477         } else {        /* assume x > 1.28 */
478             r = pr6[12]; s = ps6[12];
479             for (i = 11; i >= 0; i--) {
480                 r = z*r + pr6[i];
481                 s = z*s + ps6[i];
482             }
483         }
484         return (r/s);
485 }


486 static GENERIC qr0[12] = {      /* [16, inf] */
487   -1.24999999999999999999999999999999972972e-0001L,
488   -1.42517959554567057741439576250399159689 7e+0002L,
489   -6.31249964562597084553446025793622240721 9e+0004L,
490   -1.41137432645720838431512124369881444684 8e+0007L,
491   -1.73503421275887358141098475786078725284 2e+0009L,
492   -1.19977764751278948942182634248505528068 0e+0011L,
493   -4.59602533408165571449986040969910037364 4e+0012L,
494   -9.26252562820128410779292447703165339918 7e+0013L,
495   -8.85839472868503924534439884218066286763 9e+0014L,
496   -3.26752795368753488762374062270950597211 3e+0015L,
497   -2.66422297118631196758712934702945006201 9e+0015L,
498    3.44246406072398786958518009534450410020 4e+0014L,
499 };
500 static GENERIC qs0[11] = {
501    1.0e0L,
502    1.14072961393653646193151661000318568788 1e+0003L,
503    5.05666551044229935100919818649008580358 0e+0005L,
504    1.13204176382564278794394165052271819911 5e+0008L,
505    1.39457011187258160639262067821424647976 7e+0010L,
506    9.67794521815226478953443107956374437842 1e+0011L,
507    3.73114032785153682822514305889634850209 6e+0013L,
508    7.61278595106486929172284668102088167641 0e+0014L,
509    7.47607701640676489173019100481186397594 0e+0015L,
```

```
513    2.95124648261359203542150342710039383170 9e+0016L,
514    3.10836180369181171113685458707430203490 1e+0016L,
515 };
516 static GENERIC qr1[12] = {      /* [8, 16 ] */
517   -1.24999999999999999999794901038343381815 7e-0001L,
518   -9.05121516639382264063675224489512412693 4e+0001L,
519   -2.62078270342814883767117903190420830394 7e+0004L,
520   -3.97557126155350445776617797450878579088 4e+0006L,
521   -3.47902933075931130627007221807407499409 0e+0008L,
522   -1.82395500812426857303621674618623982908 9e+0010L,
523   -5.76593269711180137576515602922156866443 5e+0011L,
524   -1.07984368079874259295400219241793477911 4e+0013L,
525   -1.14689363050459273908220576461158133289 7e+0014L,
526   -6.36701605968389846493610444728288070418 2e+0014L,
527   -1.58310904196121349046445911190348420909 8e+0015L,
528   -1.23014955576424247310312865013579563941 2e+0015L,
529 };
530 static GENERIC qs1[14] = {
531    1.0e0L,
532    7.24683150811505811243857984777801445843 2e+0002L,
533    2.10085418443916851839938378630692703761 1e+0005L,
534    3.19263641883795150743018828594099423512 2e+0007L,
535    2.80155844338335467453844346112443421615 2e+0009L,
536    1.47502699766437373929348392725065346748 7e+0011L,
537    4.69448682491395460855236382179992714531 8e+0012L,
538    8.89035010091920025083843870960154733402 1e+0013L,
539    9.62684442908290514487470106876046975206 7e+0014L,
540    5.54111074460046077352826386268752164214 0e+0015L,
541    1.48650049478945255672747032923212309656 3e+0016L,
542    1.41584010484595940036543077373209389921 0e+0016L,
543    1.78086609524151741808131256723968233648 3e+0015L,
544   -2.35923091738488935788763154407999012949 4e+0014L,
545 };
546 static GENERIC qr2[12] = {      /* [5, 8] */
547   -1.24999999999999953193774436252772718161 4e-0001L,
548   -4.94437389735696977483937597723924157396 6e+0001L,
549   -7.72844917543346528531426165007845047390 9e+0003L,
550   -6.26257432961275234633690143465122070590 3e+0005L,
551   -2.90094822022094330602723521742438067273 2e+0007L,
552   -7.98871964763419277046391715756287411953 5e+0008L,
553   -1.31822817192718138954776002662635701237 5e+0010L,
554   -1.28243977398302924530926327194542492819 6e+0011L,
555   -7.05092557082781804018614994025791884513 8e+0011L,
556   -2.02175188257387199000420561687420268442 9e+0012L,
557   -2.59293996240066855238433390057381263565 8e+0012L,
558   -1.03826710951889126284060151493297285032 6e+0012L,
559 };
560 static GENERIC qs2[14] = {
561    1.0e0L,
562    3.96135849288557000320278402289424895211 6e+0002L,
563    6.20578873886470188282875263458651092696 8e+0004L,
564    5.04571560393267028655067381301176440674 9e+0006L,
565    2.34924861136265832353334338943096875142 9e+0008L,
566    6.52024452441582863591768355372188006391 1e+0009L,
567    1.08911121122350771933706715988628188772 2e+0011L,
568    1.08040600090535986795877940941490301861 0e+0012L,
569    6.13564528089551470351415468062376956214 8e+0012L,
570    1.86243304024662587424586715136864366821 5e+0013L,
571    2.66778080578664888884077788702193708994 e+0013L,
572    1.39440110728908777476530071180931311282 4e+0013L,
573    1.09324750061632037556289829715672244548 4e+0012L,
574   -7.22887553037892872282660421649149378077 5e+0010L,
575 };
576 static GENERIC qr3[13] = {      /* [3.5 5] */
577   -1.24999999999947306774842037957848166107 5e-0001L,
578   -3.04454904863528935191357432480325097799 8e+0001L,
```



```
579   -2.89008114064976907849669300352468144086 9e+0003L,
580   -1.40492245681720223587934327533052910768 4e+0005L,
581   -3.86274661438557344351817740361734928186 9e+0006L,
582   -6.25751730911024904920113370891115504768 9e+0007L,
583   -6.03145133092083991698707978272732347752 0e+0008L,
584   -3.41154240517383061145402576575585438234 6e+0009L,
585   -1.08939247814972667213301449872302152609 9e+0010L,
586   -1.82493407842021094129014090341595678272 6e+0010L,
587   -1.40078027830435871042348107048693953113 9e+0010L,
588   -3.71648413606491736392663571674377109209 3e+0009L,
589   -1.39759107529642552997043489095490433158 0e+0008L,
590 };
591 static GENERIC qs3[13] = {
592    1.0e0L,
593    2.44149861390496204939100018701494585804 2e+0002L,
594    2.32618888207237071150016422234151433704 3e+0004L,
595    1.13713821312123133849497710465923957816 5e+0006L,
596    3.15291807073566272872299845260536425351 7e+0007L,
597    5.17287799342650725931427048844401359510 8e+0008L,
598    5.08308643973166980745596107885647077411 5e+0009L,
599    2.96184273206643412311932552113947690994 1e+0010L,
600    9.91218586686244073582978185608135315139 0e+0010L,
601    1.79356056125162223443056418156729798359 8e+0011L,
602    1.57709011934122812252526510849794040307 3e+0011L,
603    5.50991030678016619433388999998546368163 6e+0010L,
604    4.76169113407887449120232018151793675814 1e+0009L,
605 };
606 static GENERIC qr4[13] = {      /* [2.5 3.5] */
607   -1.24999999992856773433974504349070534083 5e-0001L,
608   -1.96720174873141906305160162443556552848 1e+0001L,
609   -1.18632914671456223640709974061552817070 7e+0003L,
610   -3.60773695922294181035630149115245793406 0e+0004L,
611   -6.11920071797810490493282846857519426712 5e+0005L,
612   -6.03784778115835822667030507865220558638 4e+0006L,
613   -3.50355815333614035970053672039356598474 0e+0007L,
614   -1.18019647826822571875721852374678730977 3e+0008L,
615   -2.22186023208513491584142636350516968052 8e+0008L,
616   -2.17337250545274758529617676170174623676 0e+0008L,
617   -9.64936486506123755851773053950656801396 3e+0007L,
618   -1.46542922784793303454603964009486265038 5e+0007L,
619   -3.08300319792026208517058186624666338060 7e+0005L,
620 };
621 static GENERIC qs4[13] = {      /* [2.5 3.5] */
622    1.0e0L,
623    1.57962077373225914275261414213998685405 5e+0002L,
624    9.58137222032913873320387950375368505496 8e+0003L,
625    2.93959867237910809577611413101082588530 8e+0005L,
626    5.05218304931454221863034181869258844816 8e+0006L,
627    5.08349769559520663943383932633897198014 9e+0007L,
628    3.03638536180055338804971901400509920651 6e+0008L,
629    1.06782648145275340991056378516166149213 7e+0009L,
630    2.14564412555711804472074177512531966927 2e+0009L,
631    2.32411561595971994936394667349155221679 9e+0009L,
632    1.22326296211207075796695985561984701114 6e+0009L,
633    2.56976555331849542373847858594711027070 9e+0008L,
634    1.35474474429922712789790578773263656550 4e+0007L,
635 };
636 static GENERIC qr5[13] = {      /* [1.777.., 2.5] */
637   -1.24999999593663969763768042817457606997 1e-0001L,
638   -1.26084605537131145348589192342648906831 5e+0001L,
639   -4.77239846754446748080117433029014157889 5e+0002L,
640   -8.93985259999029848661376083399649059972 4e+0003L,
641   -9.18407078714954205097954222644613424319 7e+0004L,
642   -5.40603894501827445836263789773928043517 1e+0005L,
643   -1.84589654470519026101865372867817108441 8e+0006L,
644   -3.61361699068080950187866757065330807154 7e+0006L,
```

```
  -3.90878297813569325225255772041434862377 9e+0006L,
  -2.17371102251732392710913817058844276817 6e+0006L,
  -5.43125313067991848583640854900785624449 5e+0005L,
  -4.59109854645268451008259158727594076595 9e+0004L,
  -5.24471136416820780683552005779222964657 8e+0002L,
};
static GENERIC qs5[13] = {      /* [1.777.., 2.5] */
   1.0e0L,
   1.01453621085129087835089275097247486144 7e+0002L,
   3.87554751068713531406443416009613968107 6e+0003L,
   7.36191312267007981495525928199561773258 0e+0004L,
   7.72028894421877112658108653958552931463 6e+0005L,
   4.68152955444675249640443143360830655803 8e+0006L,
   1.66788262194050392545503125230836774582 0e+0007L,
   3.46940315761399881888272620855305156241 e+0007L,
   4.09699204796421071186708938471994786301 9e+0007L,
   2.59680475582921744931153073595956063055 4e+0007L,
   7.98393377469788923815446506401941076384 5e+0006L,
   9.81813381697990081908724242528075793815 2e+0005L,
   3.06108393086869439601354153567074544356 0e+0004L,
};

static GENERIC qr6[13] = {      /* [1.28, 1.777..] */
  -1.24999881577289001807137282824929082771 e-0001L,
  -7.99827351005311075961081059411953361928 2e+0000L,
  -1.87248195533517254336908961777156563271 9e+0002L,
  -2.12211678672630080507987400330379964681 2e+0003L,
  -1.29385028583952928250317826348477347845 7e+0004L,
  -4.44502474226631618103335419226252935609 3e+0004L,
  -8.73016137833435776766834446735650534707 0e+0004L,
  -9.70622289517207844280144497250531505473 6e+0004L,
  -5.89632551825985827016553151361819532104 1e+0004L,
  -1.82317203436810882227642082707466883223 3e+0004L,
  -2.50930417863505592663883304033747238717 5e+0003L,
  -1.15660896571577923731676982894172996409 9e+0002L,
  -7.02800578965073139688734682639778521044 2e-0001L,
};
static GENERIC qs6[13] = {      /* [1.28, 1.777..] */
   1.0e0L,
   6.45721108505806484560126132127772107590 0e+0001L,
   1.53400521658801121034282455513600868295 0e+0003L,
   1.77721799917644178259335766046237909717 1e+0004L,
   1.11837265264246946809108481026323119969 6e+0005L,
   4.01524233858461813142365748386473605294 e+0005L,
   8.37708104551709864544861651438828049767 3e+0005L,
   1.01149502000801035257539800960416428733 7e+0006L,
   6.88672207529043056865222787520020895597 0e+0005L,
   2.50473518994802147204715714861317195653 7e+0005L,
   4.40813892017104484694100184435200981706 2e+0004L,
   3.10557217807211514567305872285364085488 4e+0003L,
   5.58829482111891611343739650457381703367 8e+0001L,
};
static GENERIC qzero(x)
GENERIC x;
{
        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];
            }
        } 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];
            }
        }
        return (t*(r/s));
}
```

```
**********************************************************
   28298 Thu Oct  9 19:48:54 2014
new/usr/src/lib/libm/common/LD/j1l.c
libm - cstyle fixes
minor changes
patch07 - removed dead code with mtsk.h
patch06 - libm: fixed compilation issues after updates
libm fixes from richlowe - richlowe.net/webrevs/il_keith
patch01 - 693 import Sun Devpro Math Library
libm - cstyle fixes
minor changes
patch07 - removed dead code with mtsk.h
patch06 - libm: fixed compilation issues after updates
libm fixes from richlowe - richlowe.net/webrevs/il_keith
patch01 - 693 import Sun Devpro Math Library
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" 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.
 */

/*
 * 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 "libm_synonyms.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 = 5.64189583547756286948079451560772585844 1e-0001L,
tpi     = 0.63661977236758134307553505349005744 8L;

static GENERIC pone(), qone();
static GENERIC r0[7] = {
  -6.24999999999999999999999999999999627320e-0002L,
   1.94060672719404171620538461849464 1565464e-0003L,
  -3.00563042315573701856481469986459043883e-0005L,
   2.34558621940391866746834104736957216 9358e-0007L,
  -9.97680928588525358752901010913333666 9724e-0010L,
   2.21874325836362394607895878377510747 3381e-0012L,
  -2.07107965621870060476765092410357804 6280e-0015L,
};
static GENERIC s0[7] = {
   1.0e0L,
   1.06169590315619992073805127707500305 9555e-0002L,
   5.52186051311118037156695117939886 2692060e-0005L,
   1.82421436741375419352410787708497944 1407e-0007L,
   4.09895777843957683481883819803902935 3925e-0010L,
   6.04773507969966638985324009092526405 6197e-0013L,
   4.67904472887883619724792327951204703 5041e-0016L,
};

GENERIC
j1l(x) GENERIC x; {
        GENERIC z, d, s, c, ss, cc, r;
        int i, sgn;

        if (!finitel(x))
                return (one/x);
        sgn = signbitl(x);
        x = fabsl(x);
        if (x > 1.28L) {
                s = sinl(x);
                c = cosl(x);
        /*
         * 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.
         */
                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;
                } else {
                        ss = -s-c;
                        cc = cosl(x+x)/ss;
                }
        /*
         * 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)
```

```
117          */
118                 if (x > 1.0e120L)
119                                         return (invsqrtpi*cc)/sqrtl(x);
120                 d =  invsqrtpi*(pone(x)*cc-qone(x)*ss)/sqrtl(x);
121                 if (sgn == 0)
122                         return (d);
123                 else
124                         return (-d);
125         }
126         if (x <= small) {
127             if (x <= tiny) d = 0.5L*x;
128             else d =  x*(0.5L-x*x*0.125L);
129             if (sgn == 0)
130                         return (d);
131                 else
132                         return (-d);
133         }
134         z = x*x;
135             r = r0[6];
136             s = s0[6];
137             for (i = 5; i >= 0; i--) {
138                 r = r*z + r0[i];
139                 s = s*z + s0[i];
140             }
141         d = x*0.5L+x*(z*(r/s));
142         if (sgn == 0)
143                 return (d);
144         else
145                 return (-d);
146 }

148 static GENERIC u0[7] = {
149   -1.960570906462389484060557273467558703503e-0001L,
150    5.166389353148318460304315890665450006495e-0002L,
151   -2.229699464105910913337190798743451115604e-0003L,
152    3.625437034548863342715657067759078267158e-0005L,
153   -2.689902826993117212255524537353883987171e-0007L,
154    9.304570592456930912969387719010256018466e-0010L,
155   -1.234878126794286643318321347997500346131e-0012L,
156 };
157 static GENERIC v0[8] = {
158    1.0e0L,
159    1.369394302535807332517110204820556695644e-0002L,
160    9.508438148097659501433367062605935379588e-0005L,
161    4.399007309420092056052714797296467565655e-0007L,
162    1.488083087443756398305819693177715000787e-0009L,
163    3.751609832625793536245746965768587624922e-0012L,
164    6.680926434086257291872903276124244131448e-0015L,
165    6.676602383908906988160099057991121446058e-0018L,
166 };

168 GENERIC
169 y1l(x) GENERIC x; {
170         GENERIC z, s, c, ss, cc, u, v;
171         int i;

173         if (isnanl(x))
174                 return (x+x);
175         if (x <= zero) {
176                 if (x == zero)
177                     return (-one/zero);
178                 else
179                     return (zero/zero);
180         }
181         if (x > 1.28L) {
182                 if (!finitel(x))
```



```
183                         return (zero);
184                 s = sinl(x);
185                 c = cosl(x);
186         /*
187          * j1(x) = sqrt(2/(pi*x))*(p1(x)*cos(x0)-q1(x)*sin(x0))
188          * where x0 = x-3pi/4
189          *      Better formula:
190          *              cos(x0) = cos(x)cos(3pi/4)+sin(x)sin(3pi/4)
191          *                      =  1/sqrt(2) * (sin(x) - cos(x))
192          *              sin(x0) = sin(x)cos(3pi/4)-cos(x)sin(3pi/4)
193          *                      = -1/sqrt(2) * (cos(x) + sin(x))
194          * To avoid cancellation, use
195          *              sin(x) +- cos(x) = -cos(2x)/(sin(x) -+ cos(x))
196          * to compute the worse one.
197          */
198                 if (x > 1.0e2450L) {    /* x+x may overflow */
199                         ss = -s-c;
200                         cc =  s-c;
201                 } else if (signbitl(s) != signbitl(c)) {
202                         cc = s - c;
203                         ss = cosl(x+x)/cc;
204                 } else {
205                         ss = -s-c;
206                         cc = cosl(x+x)/ss;
207                 }
208         /*
209          * j1(x) = 1/sqrt(pi*x) * (P(1,x)*cc - Q(1,x)*ss)
210          * y1(x) = 1/sqrt(pi*x) * (P(1,x)*ss + Q(1,x)*cc)
211          */
212                 if (x > 1.0e91L)
213                         return (invsqrtpi*ss)/sqrtl(x);
214         return (invsqrtpi*(pone(x)*ss+qone(x)*cc)/sqrtl(x));
215         }
216         if (x <= tiny) {
217             return (-tpi/x);
218         }
219         z = x*x;
220             u = u0[6]; v = v0[6]+z*v0[7];
221             for (i = 5; i >= 0; i--) {
222                 u = u*z + u0[i];
223                 v = v*z + v0[i];
224             }
225         return (x*(u/v) + tpi*(j1l(x)*logl(x)-one/x));
226 }

228 static GENERIC pr0[12] = {
229    1.00000000000000000000000000000000000267e+0000L,
230    1.06071787504589145560218084327675800303 5e+0003L,
231    4.34434754289212702444668771218110585233 5e+0005L,
232    8.91568022072400701637792425271741045709 4e+0007L,
233    9.96950225993840606280987325756917127281 9e+0009L,
234    6.20029019313861303564651033870738631659 5e+0011L,
235    2.10597854878801511985181585442224733011 8e+0013L,
236    3.69663577278460123937173081031199836894 8e+0014L,
237    3.01591309792069468205795841253413451515 6e+0015L,
238    9.37029847133935309812327742732859272592 1e+0015L,
239    7.19034900519633596734079926507402944305 7e+0015L,
240    2.73609778624068999688039107492755251798 2e+0014L,
241 };
242 static GENERIC ps0[11] = {
243    1.0e0L,
244    1.06060068754589145560218084327675809510 7e+0003L,
245    4.34310609341697558914715390650533890096 1e+0005L,
246    8.91060586900217656658207224224433399059 e+0007L,
247    9.95912205863508788869071391762205654019 0e+0009L,
248    6.18874496723494823179248294917104184389 4e+0011L,
```

```
   2.0988639769537835064017598738019903049 07e+0013L,
   3.6728703570180631967467297514799389084 50e+0014L,
   2.9755384192468249210490115295743858888 420e+0015L,
   9.0636576599504320501868602928447987370 91e+0015L,
   6.4019533443147479167293664415088927116 91e+0015L,
};
static GENERIC pr1[12] = {
   1.0000000000000000000000236675241306609 84e+0000L,
   6.7461544199796187543548034881264529712 04e+0002L,
   1.8112107810833901548570183302961459705 02e+0005L,
   2.5330983903799242680380053290952878422 44e+0007L,
   2.0296836198053421452523385708754246007 29e+0009L,
   9.6608596621927114653010694015989299803 19e+0010L,
   2.7433962386448315199340989677166213163 16e+0012L,
   4.5530973541408543779310231702634552462 88e+0013L,
   4.2102450698522197574761698649748707203 74e+0014L,
   1.9873340562295964850766459671761698017 27e+0015L,
   4.0671200527870968938389704557513389304 62e+0015L,
   2.4865396063804063983108452649106913981 33e+0015L,
};
static GENERIC ps1[14] = {
   1.0e0L,
   6.7449825449796187543558086801968595217 82e+0002L,
   1.8104217953969667620321552904413647403 50e+0005L,
   2.5309864606443106515295837596999884355 73e+0007L,
   2.0267432760480231213602492888182902241 45e+0009L,
   9.6374619244074059352452694070526413418 36e+0010L,
   2.7323786284237664174022927970283141608 31e+0012L,
   4.5223452749605271243548443640121842784 88e+0013L,
   4.1606506683417431326853357584154698565 45e+0014L,
   1.9437302429888582082434924248924359012 11e+0015L,
   3.8802285326921279899011316185980674500 01e+0015L,
   2.1780208161611546158410001736833029997 28e+0015L,
  -8.9940626668422255515543466981716006341 73e+0013L,
   1.3685203685088512534957648069346195749 90e+0013L,
};
static GENERIC pr2[12] = {
   1.0000000000000000006938651621840396237 282e+0000L,
   3.6584162918504049814071010770379481446 98e+0002L,
   5.2670737721703565477097946706028124475 37e+0004L,
   3.9120121012268374630149252107358946204 42e+0006L,
   1.6512956489741039571938749287141807656 25e+0008L,
   4.1149011444807976099724849981421467834 99e+0009L,
   6.0925243097660366815429805725263351476 72e+0010L,
   5.2639131780712826167192499690741345705 77e+0011L,
   2.5384085811243242233673410205380813309 94e+0012L,
   6.2886079293602910278951269830153656776 48e+0012L,
   6.8483300482111484190470550753865259452 80e+0012L,
   2.2903096468388679414231781639914232446 90e+0012L,
};
static GENERIC ps2[14] = {
   1.0e0L,
   3.6572444168504050864594101657623198618 56e+0002L,
   5.2628023584250232439923870758612373063 12e+0004L,
   3.9058968139599196481362958616614838483 64e+0006L,
   1.6467919077914612207426948421082027727 63e+0008L,
   4.0961328030642560222249541202082014373 44e+0009L,
   6.0466651959159504475442944573068023675 9e+0010L,
   5.1980617397819913134140522123286532951 68e+0011L,
   2.4842338518143339664015276264212542797 96e+0012L,
   6.0478688069253158793396515394343152559 40e+0012L,
   6.3331038312540916525016425672941018133 54e+0012L,
   1.8751430987542849944676099369246850249 68e+0012L,
  -5.2383309205633926929654127625088136015 34e+0010L,
   4.6568886094393647254277891983837792599 57e+0009L,
};
```



```
static GENERIC pr3[13] = {
   1.0000000000000093368873186805613784289 7e+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,
   4.5756199941271607806407058776741643639 6e+0010L,
   1.2284156512116391606202844416905035508 42e+0010L,
   7.2421703498755630534360505321531128820 72e+0008L,
};
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,
   4.0429626592715679487356768346093488422 92e+0010L,
   6.0281684626466945100838472229684444021 61e+0010L,
   4.1184102267946414138338876065800852811 11e+0010L,
   9.9187357362970384307441612533382022302 63e+0009L,
   4.0929671982380980232191244874371303320 38e+0008L,
};
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,
};
static GENERIC ps4[13] = {
   1.0e0L,
   1.4363799933845323716704933195918473623 04e+0002L,
   7.8946471547854306780610538488474366594 99e+0003L,
   2.1846597533920975290089817415508785861 74e+0005L,
   3.3661090833054651768035137381470494993 61e+0006L,
   3.0119115459689968176976658666587226343 186e+0007L,
   1.5822629137796898513167601484594148953 01e+0008L,
   4.8192688094949379192179385895301382017 70e+0008L,
   8.2013557629904506797028371234325271548 30e+0008L,
   7.2682320939825109374174464212823414252 12e+0008L,
   2.9509119090155729332621313239340364804 62e+0008L,
   4.2428399243059344230108589665406212193 96e+0007L,
   1.0643876204450907791821176663304051868 66e+0006L,
};
static GENERIC pr5[13] = {
   1.0000000001024348052411714272538473538 61e+0000L,
   9.1293322570836292590605022490259632348 21e+0001L,
   3.1322384835869530375761193775045571914 13e+0003L,
   5.3297825282693079712789431224541711078 61e+0004L,
   4.9884601571841177906928730021030529441 45e+0005L,
```

```
   2.68660207161578681614701033425604746937 8e+0006L,
   8.44541852602896119770379980870126830183 1e+0006L,
   1.53657535864614115747572588990790082739 0e+0007L,
   1.56840581823652382179686277058654481194 5e+0007L,
   8.45087623988877010238761866736230217354 7e+0006L,
   2.15441490013956732842402682716320344607 7e+0006L,
   2.10565692656504389888846025480806235220 5e+0005L,
   4.73916501102339650702213430373686281297 5e+0003L,
};
static GENERIC ps5[13] = {
   1.0e0L,
   9.11761350959532747650915267339470384779 3e+0001L,
   3.12169797248401563930127922928177079514 7e+0003L,
   5.29444722273589356804091187383457644025 5e+0004L,
   4.93036888219277233579825668411088788280 7e+0005L,
   2.63485468564116529830216743579835743776 8e+0006L,
   8.18546277540032639355589615703181828091 8e+0006L,
   1.46241742308021519260966864266303066708 6e+0007L,
   1.45062499398585167598286084415395489601 5e+0007L,
   7.46046764756199528321908656716200611386 4e+0006L,
   1.75421098140561247886922714257905633896 5e+0006L,
   1.46328672115527197152626491452474669959 6e+0005L,
   2.15589472579670201534121111657982703945 9e+0003L,
};
static GENERIC pr6[13] = {
   1.00000000356485554674173592031574315712 9e+0000L,
   5.73400393486254045811942350990951028836 6e+0001L,
   1.20957249193585048608655969229179688797 6e+0003L,
   1.24339839142228124793367477916366028683 8e+0004L,
   6.93099675518143793725822099860170827878 7e+0004L,
   2.19806765953275759864672224996676762009 9e+0005L,
   4.03365943271205863393317911582057685845 5e+0005L,
   4.25775965721900802701604720657457435867 8e+0005L,
   2.51191739587600434948072127744576391638 9e+0005L,
   7.81375615307062365417873165138188195355 2e+0004L,
   1.15206917338112788138558809290586435289 1e+0004L,
   6.54858078280408855377781603755152339808 2e+0002L,
   8.66872537011690613232754276612793849688 0e+0000L,
};
static GENERIC ps6[13] = {
   1.0e0L,
   5.72228523635711456649922152573628620518 4e+0001L,
   1.20301084287831793544458295062033957050 6e+0003L,
   1.23005833578583550155825502172435371208 e+0004L,
   6.80099855060786128886530043864808989441 2e+0004L,
   2.13076782959930426298776934753685088592 1e+0005L,
   3.84048346664391668175993697299215531002 6e+0005L,
   3.94743237345922554286181914810808116039 3e+0005L,
   2.23781623939308111148158843445783852673 8e+0005L,
   6.54582049512441972339894627379092154077 4e+0004L,
   8.72956363032089274150072621327883473719 6e+0003L,
   4.13076266029189475345017479419699881370 9e+0002L,
   3.48036889867268464513033578601507559559 8e+0000L,
};
static GENERIC sixteen = 16.0L;
static GENERIC eight   = 8.0L;
static GENERIC huge    = 1.0e30L;

static GENERIC pone(x)
GENERIC x;
{
        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];
            }
        } 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];
            }
        }
        return (r/s);
}


static GENERIC qr0[12] = {
   3.74999999999999999999999999999971033e-0001L,
   4.25672603523705060160768227743309426222 6e+0002L,
   1.87597649081287848919240997894540106606 6e+0005L,
   4.17031426804804191427360368031774559279 0e+0007L,
   5.09275013254385581729345111897455574655 1e+0009L,
   3.49474967627848865410350579579413948340 4e+0011L,
   1.32706214825743731699766781709669417370 9e+0013L,
   2.64899313827342722690750374206655115049 0e+0014L,
   2.51169566590954741222243049447399812768 4e+0015L,
   9.27469450666228904322431049916470230609 6e+0015L,
   8.15090417066366382933132030291179289200 2e+0015L,
  -5.00191873370766235577203782962038876512 2e+0014L,
};
static GENERIC qs0[11] = {
   1.0e0L,
   1.13540038022988016042871527398215576009 3e+0003L,
   5.00570118387712616432676554551659074436 0e+0005L,
   1.11344420011371216798433760393304010298 7e+0008L,
```

```
513     1.36107481992522306277871756569909347112 4e+0010L,
514     9.35575098580284948443893390532598280965 3e+0011L,
515     3.56346278600898882500396554385799808482 8e+0013L,
516     7.15514511390009416364872686380380291045 4e+0014L,
517     6.87126683583447275805559013851843654113 e+0015L,
518     2.62203089922673671264497498815734523409 2e+0016L,
519     2.60291272917287633065007702170613970774 6e+0016L,
520 };
521 static GENERIC qr1[12] = {
522     3.74999999999999999999776245820728440580 6e-0001L,
523     2.69788399888170683992925551749818998048 5e+0002L,
524     7.75519592578102848938693887047383441101 9e+0004L,
525     1.16677776210401777719821107289552896835 5e+0007L,
526     1.01150477298432116832001008452026106936 2e+0009L,
527     5.24600770357415685357775457172020555001 0e+0010L,
528     1.63769254988559268316611655169126653764 7e+0012L,
529     3.02230362369818566991299031092503938249 5e+0013L,
530     3.15476992729065568484610703026590998794 6e+0014L,
531     1.71581991344155477008973093480812336092 1e+0015L,
532     4.16504435575973262227353444513173618851 0e+0015L,
533     3.15138142087417470564310038170808628759 6e+0015L,
534 };
535 static GENERIC qs1[14] = {
536     1.0e0L,
537     7.19709170535121823978563317240827698282 8e+0002L,
538     2.07001279959954868554488304129760986105 5e+0005L,
539     3.11701481531765622187184015277845875451 6e+0007L,
540     2.70571967890255497486332587702590297172 7e+0009L,
541     1.40611361472734572692506068475086726409 8e+0011L,
542     4.40377753606713132036300597863167481735 9e+0012L,
543     8.17072569020932228306149938670316724289 4e+0013L,
544     8.60945884497549528922779412696443121056 6e+0014L,
545     4.76676636701547348125728060069495292020 4e+0015L,
546     1.20228658794334219486355794088811564165 0e+0016L,
547     1.01247432830620090952506393606175602412 0e+0016L,
548     6.18355202267891785827322287961582407070 3e+0014L,
549   -9.75673154855822699757373740098822572274 0e+0013L,
550 };
551 static GENERIC qr2[12] = {
552     3.74999999999999948124564726222699429318 9e-0001L,
553     1.47136680728977135449118114016735902673 5e+0002L,
554     2.27943248676844822014208096284352695125 0e+0004L,
555     1.82894304852377122516380404335695828589 3e+0006L,
556     8.37982838864782313583222059641772501083 7e+0007L,
557     2.27981402933504402458539367127837802205 3e+0009L,
558     3.71165395225711812083281778527146644142 0e+0010L,
559     3.55765091451855454991673057255310504806 8e+0011L,
560     1.92458348314609589625977432949893416065 0e+0012L,
561     5.42438625606373639075956708829188714027 8e+0012L,
562     6.83932562124177678620650970467174684173 7e+0012L,
563     2.70216956314400116629168645230543631397 1e+0012L,
564 };
565 static GENERIC qs2[14] = {
566     1.0e0L,
567     3.92637919443938813570321193389520319108 9e+0002L,
568     6.08914880410659829748833606300760931227 6e+0004L,
569     4.89354616297327858371137635604161415064 5e+0006L,
570     2.24757119114497845046388801813832219404 e+0008L,
571     6.13763566335017775129046933420075787264 5e+0009L,
572     1.00511501978410285642449351952499895367 8e+0011L,
573     9.72566446201450383286015138460467724062 0e+0011L,
574     5.34552510048551111614863419284443463607 2e+0012L,
575     1.54994400739894669172086273817395699477 9e+0013L,
576     2.06714841178952625710302124163264760362 e+0013L,
577     9.40156540264196361129511948724259546230 1e+0012L,
578     3.54821708862239827474883728776970937438 5e+0011L,
```



```
579   -2.93447034171904712007650993843241735236 5e+0010L,
580 };
581 static GENERIC qr3[13] = {
582     3.74999999999941272408457983329745147209 1e-0001L,
583     9.05847858029170621242297849293843558252 7e+0001L,
584     8.52405603316103875046108366671172438117 1e+0003L,
585     4.10596715862910942775343456922363101473 0e+0005L,
586     1.11832660337853134825978309197262333365 7e+0007L,
587     1.79463668340357891852806490471413232934 3e+0008L,
588     1.71431415746363595955613323600436889672 4e+0009L,
589     9.62209203223608484657206725726766145603 0e+0009L,
590     3.05775952448585915995776285878076835502 0e+0010L,
591     5.12930678075479853160962145441593889002 0e+0010L,
592     3.99912200279496107068063619434631604135 2e+0010L,
593     1.12229845464349348598972156435810034538 8e+0010L,
594     5.60398198764598970966883096852236258222 1e+0008L,
595 };
596 static GENERIC qs3[13] = {
597     1.0e0L,
598     2.41832866307657816983615517005363441992 2e+0002L,
599     2.27962020590012104258752354128127287552 0e+0004L,
600     1.10098422258572952147012901499221709279 4e+0006L,
601     3.01074322367924709100426251628665451628 2e+0007L,
602     4.86092554282736781728961926521559943399 6e+0008L,
603     4.68666811103534869198271586430783958124 3e+0009L,
604     2.66870178840510201742721470594673089407 4e+0010L,
605     8.67739574610680264039058094483665058490 3e+0010L,
606     1.51193645557495179065849879594510664303 6e+0011L,
607     1.26084560443262347800201869687360835309 3e+0011L,
608     4.05269227841985385391144023160086458980 5e+0010L,
609     2.96551651921222606498326782224332969472 9e+0009L,
610 };
611 static GENERIC qr4[13] = {
612     3.74999999991923416415466975440123072618 e-0001L,
613     5.84421858077681986479116825348505510185 8e+0001L,
614     3.48927351409291298267566941137143567022 0e+0003L,
615     1.05052363777457568450966343001899547959 4e+0005L,
616     1.76454917205970156550071731979278011528 9e+0006L,
617     1.72553243884413379502806310268149737115 4e+0007L,
618     9.93811484735977853996514024759017633487 4e+0007L,
619     3.33171080818459554539688377020077284231 4e+0008L,
620     6.27197055764188151160956044487279728269 8e+0008L,
621     6.18852979867735707502077492390373791328 5e+0008L,
622     2.82190530274284997450998216787788501162 9e+0008L,
623     4.61546735864691197677329025698432981489 6e+0007L,
624     1.34814060873154646739668580269338069327 5e+0006L,
625 };
626 static GENERIC qs4[13] = {
627     1.0e0L,
628     1.56119266311234518526141829638990213337 2e+0002L,
629     9.34667803114409827054722542312421308307 2e+0003L,
630     2.82585124648229354783802384760170475159 0e+0005L,
631     4.77657271162215609171090289112491155629 3e+0006L,
632     4.71510695371713540297793804800626785930 2e+0007L,
633     2.75396235089431131643965222761120903519 3e+0008L,
634     9.42850143461546320776896478750041157522 3e+0008L,
635     1.83265085877520678708823689645414157261 7e+0009L,
636     1.90169737893974322694892087429659524225 7e+0009L,
637     9.43332222685429378062718859922638081272 5e+0008L,
638     1.80852054060867160868028452079885858737 0e+0008L,
639     7.98334233173666275315721744691946239800 8e+0006L,
640 };
641 static GENERIC qr5[13] = {
642       3.74999999533136443702898885051519044671 9e-0001L,
643       3.73935638176655988267751459304162754791 1e+0001L,
644       1.39956250062941352935526546291281980255 1e+0003L,
```

```
645       2.59415405309894792534533221806221011753e+0004L,
646       2.64014987929740864039416397939459431837 1e+0005L,
647       1.54247185487319914203188909359144939 7995e+0006L,
648       5.24227286897205337406757209899233542 5895e+0006L,
649       1.02583448776941022132963307142604483 9935e+0007L,
650       1.11655392423944894014223057906012420 9622e+0007L,
651       6.31807606559591017637491630352588465 3514e+0006L,
652       1.64121808616864040852763973591551288 1785e+0006L,
653       1.52236979352917864416881388291213470 6444e+0005L,
654       2.52653054106229720091418006020866958 4055e+0003L,
655 };
656 static GENERIC qs5[13] = {
657       1.0e0L,
658       9.99896073593507538039754565901628750 6660e+0001L,
659       3.75876741784204374268647506054041673 7562e+0003L,
660       7.01365280695230652012195974251978078 1653e+0004L,
661       7.20894980881861509924652961621173044 6850e+0005L,
662       4.27275392710961445541783618607220200 9252e+0006L,
663       1.48252441135647069933612981411102543 4703e+0007L,
664       2.98875036666567823342527923762770080 3473e+0007L,
665       3.39695789026108049269470915055361918 5065e+0007L,
666       2.05065248773859300411157809115630454 0386e+0007L,
667       5.90050412081173254761651155594627945 1316e+0006L,
668       6.56339140926016089702449808227318346 8347e+0005L,
669       1.69262984501279020534896673147718704 1419e+0004L,
670 };
671 static GENERIC qr6[13] = {
672     3.74999986151666413315756687085897542 1296e-0001L,
673     2.36786375674776486312079743159947346 8918e+0001L,
674     5.47671580211497624888206732563079314 3777e+0002L,
675     6.14319035786984289402501294544409617 0251e+0003L,
676     3.71625053467799785051373359514046385 1730e+0004L,
677     1.27088346382387675213832690502287565 7430e+0005L,
678     2.49530144963681448164637166542908380 1388e+0005L,
679     2.78957898821295224834048629625439860 1942e+0005L,
680     1.71824794691110905593181908713739732 4634e+0005L,
681     5.45897321401166571433032673220410636 4229e+0004L,
682     7.91210268668794878604894333975959665 2813e+0003L,
683     4.07796100616086693572203071514908793 8091e+0002L,
684     3.76520697277024508555105723788252851 0428e+0000L,
685 };
686 static GENERIC qs6[13] = {
687     1.0e0L,
688     6.34164653294051730564189385267392680 9601e+0001L,
689     1.47705827741404079093259753792067102 5359e+0003L,
690     1.67440656403104449143604425339353648 7604e+0004L,
691     1.02851650136975594989505080690899465 0768e+0005L,
692     3.59362004253288529508746350773328543 4207e+0005L,
693     7.26792499138102091518587339945372479 9625e+0005L,
694     8.46227751076881839996119142620500608 3088e+0005L,
695     5.51439989223089216337361189564550025 0514e+0005L,
696     1.89808424100925935354062027293218810 2299e+0005L,
697     3.10294124211773901572198412308102625 3068e+0004L,
698     1.95897118443146690768144065018142108 6143e+0003L,
699     2.87885335731049508718172161388945512 1867e+0001L,
700 };
701 static GENERIC qone(x)
702 GENERIC x;
703 {
704         GENERIC s, r, t, z;
705         int i;
706         if (x > huge)
707                 return (0.375L/x);
708         t = one/x; z = t*t;
709         if (x > sixteen) {
710             r = z*qr0[11]+qr0[10]; s = qs0[10];
```



```
711             for (i = 9; i >= 0; i--) {
712                 r = z*r + qr0[i];
713                 s = z*s + qs0[i];
714             }
715         } else if (x > eight) {
716             r = qr1[11]; s = qs1[11]+z*(qs1[12]+z*qs1[13]);
717             for (i = 10; i >= 0; i--) {
718                 r = z*r + qr1[i];
719                 s = z*s + qs1[i];
720             }
721         } else if (x > five) {  /* x > 5.0 */
722             r = qr2[11]; s = qs2[11]+z*(qs2[12]+z*qs2[13]);
723             for (i = 10; i >= 0; i--) {
724                 r = z*r + qr2[i];
725                 s = z*s + qs2[i];
726             }
727         } else if (x > 3.5L) {
728             r = qr3[12]; s = qs3[12];
729             for (i = 11; i >= 0; i--) {
730                 r = z*r + qr3[i];
731                 s = z*s + qs3[i];
732             }
733         } else if (x > 2.5L) {
734             r = qr4[12]; s = qs4[12];
735             for (i = 11; i >= 0; i--) {
736                 r = z*r + qr4[i];
737                 s = z*s + qs4[i];
738             }
739         } else if (x > (1.0L/0.5625L)) {
740             r = qr5[12]; s = qs5[12];
741             for (i = 11; i >= 0; i--) {
742                 r = z*r + qr5[i];
743                 s = z*s + qs5[i];
744             }
745         } else {                /* assume x > 1.28 */
746             r = qr6[12]; s = qs6[12];
747             for (i = 11; i >= 0; i--) {
748                 r = z*r + qr6[i];
749                 s = z*s + qs6[i];
750             }
751         }
752         return (t*(r/s));
753 }
```

```
**********************************************************
    6912 Thu Oct  9 19:48:54 2014
new/usr/src/lib/libm/common/LD/jnl.c
common/LD/jnl.c: cstyle fixes
minor changes
patch07 - removed dead code with mtsk.h
patch06 - libm: fixed compilation issues after updates
libm fixes from richlowe - richlowe.net/webrevs/il_keith
patch01 - 693 import Sun Devpro Math Library
common/LD/jnl.c: cstyle fixes
minor changes
patch07 - removed dead code with mtsk.h
patch06 - libm: fixed compilation issues after updates
libm fixes from richlowe - richlowe.net/webrevs/il_keith
patch01 - 693 import Sun Devpro Math Library
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" 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.
 */

#if defined(ELFOBJ)
#pragma weak jnl = __jnl
#pragma weak ynl = __ynl
#endif

/*
 * 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;

GENERIC
jnl(n, x) 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;
        }
        if (n == 0) return (j0l(x));
        if (n == 1) return (j1l(x));
        if (x != x) return x+x;
        if ((n&1) == 0)
                sgn = 0;                        /* even n */
        else
                sgn = signbitl(x);      /* old n  */
        x = fabsl(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) {
                    case 0: temp =  cosl(x)+sinl(x); break;
                    case 1: temp = -cosl(x)+sinl(x); break;
```

```
117                     case 2: temp = -cosl(x)-sinl(x); break;
118                     case 3: temp =  cosl(x)-sinl(x); break;
119                 }
120                 b = invsqrtpi*temp/sqrtl(x);
121             } else {
122                         a = j0l(x);
123                         b = j1l(x);
124                         for (i = 1; i < n; i++) {
125                     temp = b;
126                     b = b*((GENERIC)(i+i)/x) - a; /* avoid underflow */
127                     a = temp;
128                         }
129             }
130         } else {
131             if (x < 1e-17L) {   /* use J(n,x) = 1/n!*(x/2)^n */
132                 b = powl(0.5L*x, (GENERIC) n);
133                 if (b != zero) {
134                     for (a = one, i = 1; i <= n; i++) a *= (GENERIC)i;
135                     b = b/a;
136                 }
137             } else {
138                 /*
139                  * use backward recurrence
140                  *                      x      x^2      x^2
141                  *  J(n,x)/J(n-1,x) =  ----   ------   ------   .....
142                  *                      2n  - 2(n+1) - 2(n+2)
143                  *
144                  *                      1      1        1
145                  *  (for large x)   =  ----  ------   ------   .....
146                  *                      2n   2(n+1)   2(n+2)
147                  *                      -- - ------ - ------ -
148                  *                       x     x         x
149                  *
150                  * Let w = 2n/x and h=2/x, then the above quotient
151                  * is equal to the continued fraction:
152                  *                   1
153                  *      = -----------------------
154                  *                  1
155                  *         w - -----------------
156                  *                    1
157                  *              w+h - ---------
158                  *                    w+2h - ...
159                  *
160                  * To determine how many terms needed, let
161                  * Q(0) = w, Q(1) = w(w+h) - 1,
162                  * Q(k) = (w+k*h)*Q(k-1) - Q(k-2),
163                  * When Q(k) > 1e4      good for single
164                  * When Q(k) > 1e9      good for double
165                  * When Q(k) > 1e17     good for quaduple
166                  */
167             /* determin k */
168                 GENERIC t, v;
169                 double q0, q1, h, tmp; int k, m;
170                 w  = (n+n)/(double)x; h = 2.0/(double)x;
171                 q0 = w;  z = w+h; q1 = w*z - 1.0; k = 1;
172                 while (q1 < 1.0e17) {
173                         k += 1; z += h;
174                         tmp = z*q1 - q0;
175                         q0 = q1;
176                         q1 = tmp;
177                 }
178                 m = n+n;
179                 for (t = zero, i = 2*(n+k); i >= m; i -= 2) t = one/(i/x-t);
180                 a = t;
181                 b = one;
182                         /*
```



```
183                          * Estimate log((2/x)^n*n!) = n*log(2/x)+n*ln(n)
184                          * hence, if n*(log(2n/x)) > ...
185                          * single 8.8722839355e+01
186                          * double 7.09782712893383973096e+02
187                          * long double 1.13565234062941439494919310779707650061 7
188                          * then recurrent value may overflow and the result is
189                          * likely underflow to zero.
190                          */
191                 tmp = n;
192                 v = two/x;
193                 tmp = tmp*logl(fabsl(v*tmp));
194                 if (tmp < 1.13565234062941439494919310779707650 5e+04L) {
195                                 for (i = n-1; i > 0; i--) {
196                                 temp = b;
197                                 b = ((i+i)/x)*b - a;
198                                 a = temp;
199                                 }
200                 } else {
201                                 for (i = n-1; i > 0; i--) {
202                                 temp = b;
203                                 b = ((i+i)/x)*b - a;
204                                 a = temp;
205                         if (b > 1e1000L) {
206                                                 a /= b;
207                                                 t /= b;
208                                                 b  = 1.0;
209                                         }
210                                 }
211                 }
212                         b = (t*j0l(x)/b);
213             }
214         }
215         if (sgn == 1)
216                 return -b;
217         else
218                 return b;
219 }

221 GENERIC
222 ynl(n, x) int n; GENERIC x; {
223         int i;
224         int sign;
225         GENERIC a, b, temp = 0;

227         if (x != x)
228                 return x+x;
229         if (x <= zero) {
230                 if (x == zero)
231                         return -one/zero;
232                 else
233                         return zero/zero;
234         }
235         sign = 1;
236         if (n < 0) {
237                 n = -n;
238                 if ((n&1) == 1) sign = -1;
239         }
240         if (n == 0) return (y0l(x));
241         if (n == 1) return (sign*y1l(x));
242         if (!finitel(x)) return zero;

244         if (x > 1.0e91L) {
245                                 /*
246                                  * x >> n**2
247                                  * Jn(x) = cos(x-(2n+1)*pi/4)*sqrt(2/x*pi)
248                                  *   Yn(x) = sin(x-(2n+1)*pi/4)*sqrt(2/x*pi)
```

```
249                                 *   Let s=sin(x), c=cos(x),
250                                 * xn=x-(2n+1)*pi/4, sqt2 = sqrt(2),then
251                                 *
252                                 *         n    sin(xn)*sqt2    cos(xn)*sqt2
253                                 *      ----------------------------------
254                                 *         0     s-c             c+s
255                                 *         1    -s-c            -c+s
256                                 *         2    -s+c            -c-s
257                                 *         3     s+c             c-s
258                                 */
259                 switch (n&3) {
260                     case 0: temp =  sinl(x)-cosl(x); break;
261                     case 1: temp = -sinl(x)-cosl(x); break;
262                     case 2: temp = -sinl(x)+cosl(x); break;
263                     case 3: temp =  sinl(x)+cosl(x); break;
264                 }
265                 b = invsqrtpi*temp/sqrtl(x);
266         } else {
267                 a = y0l(x);
268                 b = y1l(x);
269                 /*
270                  * fix 1262058 and take care of non-default rounding
271                  */
272                 for (i = 1; i < n; i++) {
273                         temp = b;
274                         b *= (GENERIC) (i + i) / x;
275                         if (b <= -LDBL_MAX)
276                                 break;
277                         b -= a;
278                         a = temp;
279                 }
280         }
281         if (sign > 0)
282                 return b;
283         else
284                 return -b;
285 }
```

```
**********************************************************
    2628 Thu Oct  9 19:48:54 2014
new/usr/src/lib/libm/common/LD/tanhl.c
fix lint warnings in LD/asinhl.c and LD/tanhl.c
libm/common/R/tanf.c
libm/common/R/sinf.c
libm/common/R/sincosf.c
libm/common/R/cosf.c
libm/common/Q/tanhl.c
libm/common/Q/asinl.c
libm/common/Q/asinhl.c
tanhl.c
patch07 - removed dead code with mtsk.h
patch06 - libm: fixed compilation issues after updates
libm fixes from richlowe - richlowe.net/webrevs/il_keith
patch01 - 693 import Sun Devpro Math Library
fix lint warnings in LD/asinhl.c and LD/tanhl.c
libm/common/R/tanf.c
libm/common/R/sinf.c
libm/common/R/sincosf.c
libm/common/R/cosf.c
libm/common/Q/tanhl.c
libm/common/Q/asinl.c
libm/common/Q/asinhl.c
tanhl.c
patch07 - removed dead code with mtsk.h
patch06 - libm: fixed compilation issues after updates
libm fixes from richlowe - richlowe.net/webrevs/il_keith
patch01 - 693 import Sun Devpro Math Library
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" 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.
 */

#if defined(ELFOBJ)
#pragma weak tanhl = __tanhl
#endif

/*
 * 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,
#ifndef lint
        big = 1.0e+20L,
#endif
        threshold = 45.0L;

long double
tanhl(long double x) {
        long double t, y, z;
        int signx;
#ifndef lint
        volatile long double dummy;
#endif

        if (isnanl(x))
                return (x + x);         /* x is NaN */
        signx = signbitl(x);
        t = fabsl(x);
        z = one;
        if (t <= threshold) {
                if (t > one)
                        z = one - two / (expm1l(t + t) + two);
                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))
                return (copysignl(one, x));
        else
                return (signx ? -z + small * small : z - small * small);
        return (signx ? -z : z);
}
```

```
**********************************************************
   12633 Thu Oct  9 19:48:54 2014
new/usr/src/lib/libm/common/m9x/fenv_inlines.h
fixes for %1 +x in common/m9x/fenv_inlines.h
fix fsincos in common/m9x/fenv_inlines.h
patch05 - fixed amd64 issues with LIBM
libm fixes from richlowe - richlowe.net/webrevs/il_keith
patch01 - 693 import Sun Devpro Math Library
fixes for %1 +x in common/m9x/fenv_inlines.h
fix fsincos in common/m9x/fenv_inlines.h
patch05 - fixed amd64 issues with LIBM
libm fixes from richlowe - richlowe.net/webrevs/il_keith
patch01 - 693 import Sun Devpro Math Library
**********************************************************
```

```
/*
 * 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;
};

extern __inline__ void
__fenv_getcwsw(unsigned int *value)
{
        union fp_cwsw *u = (union fp_cwsw *)value;

        __asm__ __volatile__(
            "fstsw %0\n\t"
            "fstcw %1\n\t"
            : "=m" (u->words.cw), "=m" (u->words.sw));
}
```



```
extern __inline__ void
__fenv_setcwsw(const unsigned int *value)
{
        union fp_cwsw cwsw;
        short fenv[16];

        cwsw.cwsw = *value;

        __asm__ __volatile__(
            "fstenv %0\n\t"
            "movw   %4,%1\n\t"
            "movw   %3,%2\n\t"
            "fldenv %0\n\t"
            "fwait\n\t"
            : "=m" (fenv), "=m" (fenv[0]), "=m" (fenv[2])
            : "r" (cwsw.words.cw), "r" (cwsw.words.sw)
            /* For practical purposes, we clobber the whole FPU */
            : "cc", "st", "st(1)", "st(2)", "st(3)", "st(4)", "st(5)",
              "st(6)", "st(7)");
}

extern __inline__ void
__fenv_getmxcsr(unsigned int *value)
{
        __asm__ __volatile__("stmxcsr %0" : "=m" (*value));
}

extern __inline__ void
__fenv_setmxcsr(const unsigned int *value)
{
        __asm__ __volatile__("ldmxcsr %0" : : "m" (*value));
}

extern __inline__ long double
f2xm1(long double x)
{
        long double ret;

        __asm__ __volatile__("f2xm1" : "=t" (ret) : "0" (x) : "cc");
        return (ret);
}

extern __inline__ long double
fyl2x(long double y, long double x)
{
        long double ret;

        __asm__ __volatile__("fyl2x"
            : "=t" (ret)
            : "0" (x), "u" (y)
            : "st(1)", "cc");
        return (ret);
}

extern __inline__ long double
fptan(long double x)
{
        /*
         * fptan pushes 1.0 then the result on completion, so we want to pop
         * the FP stack twice, so we need a dummy value into which to pop it.
         */
        long double ret;
        long double dummy;

        __asm__ __volatile__("fptan"
            : "=t" (dummy), "=u" (ret)
```

```
119             : "0" (x)
120             : "cc");
121         return (ret);
122 }

124 extern __inline__ long double
125 fpatan(long double x, long double y)
126 {
127         long double ret;

129         __asm__ __volatile__("fpatan"
130             : "=t" (ret)
131             : "0" (y), "u" (x)
132             : "st(1)", "cc");
133         return (ret);
134 }

136 extern __inline__ long double
137 fxtract(long double x)
138 {
139         __asm__ __volatile__("fxtract" : "+t" (x) : : "cc");
140         return (x);
141 }

143 extern __inline__ long double
144 fprem1(long double idend, long double div)
145 {
146         __asm__ __volatile__("fprem1" : "+t" (div) : "u" (idend) : "cc");
147         return (div);
148 }

150 extern __inline__ long double
151 fprem(long double idend, long double div)
152 {
153         __asm__ __volatile__("fprem" : "+t" (div) : "u" (idend) : "cc");
154         return (div);
155 }

157 extern __inline__ long double
158 fyl2xp1(long double y, long double x)
159 {
160         long double ret;

162         __asm__ __volatile__("fyl2xp1"
163             : "=t" (ret)
164             : "0" (x), "u" (y)
165             : "st(1)", "cc");
166         return (ret);
167 }

169 extern __inline__ long double
170 fsqrt(long double x)
171 {
172         __asm__ __volatile__("fsqrt" : "+t" (x) : : "cc");
173         return (x);
174 }

176 extern __inline__ long double
177 fsincos(long double x)
178 {
179         long double dummy;

181         __asm__ __volatile__("fsincos" : "+t" (x), "=u" (dummy) : : "cc");
182         return (x);
183 }
```



```
185 extern __inline__ long double
186 frndint(long double x)
187 {
188         __asm__ __volatile__("frndint" : "+t" (x) : : "cc");
189         return (x);
190 }

192 extern __inline__ long double
193 fscale(long double x, long double y)
194 {
195         long double ret;

197         __asm__ __volatile__("fscale" : "=t" (ret) : "0" (y), "u" (x) : "cc");
198         return (ret);
199 }

201 extern __inline__ long double
202 fsin(long double x)
203 {
204         __asm__ __volatile__("fsin" : "+t" (x) : : "cc");
205         return (x);
206 }

208 extern __inline__ long double
209 fcos(long double x)
210 {
211         __asm__ __volatile__("fcos" : "+t" (x) : : "cc");
212         return (x);
213 }

215 extern __inline__ void
216 sse_cmpeqss(float *f1, float *f2, int *i1)
217 {
218         __asm__ __volatile__(
219             "cmpeqss %2, %1\n\t"
220             "movss   %1, %0"
221             : "=m" (*i1), "+x" (*f1)
222             : "x" (*f2)
223             : "cc");
224 }

226 extern __inline__ void
227 sse_cmpltss(float *f1, float *f2, int *i1)
228 {
229         __asm__ __volatile__(
230             "cmpltss %2, %1\n\t"
231             "movss   %1, %0"
232             : "=m" (*i1), "+x" (*f1)
233             : "x" (*f2)
234             : "cc");
235 }

237 extern __inline__ void
238 sse_cmpless(float *f1, float *f2, int *i1)
239 {
240         __asm__ __volatile__(
241             "cmpless %2, %1\n\t"
242             "movss   %1, %0"
243             : "=m" (*i1), "+x" (*f1)
244             : "x" (*f2)
245             : "cc");
246 }

248 extern __inline__ void
249 sse_cmpunordss(float *f1, float *f2, int *i1)
250 {
```

```
251         __asm__ __volatile__(
252             "cmpunordss %2, %1\n\t"
253             "movss      %1, %0"
254             : "=m" (*i1), "+x" (*f1)
255             : "x" (*f2)
256             : "cc");
257 }

259 extern __inline__ void
260 sse_minss(float *f1, float *f2, float *f3)
261 {
262         __asm__ __volatile__(
263             "minss %2, %1\n\t"
264             "movss %1, %0"
265             : "=m" (*f3), "+x" (*f1)
266             : "x" (*f2));
267 }

269 extern __inline__ void
270 sse_maxss(float *f1, float *f2, float *f3)
271 {
272         __asm__ __volatile__(
273             "maxss %2, %1\n\t"
274             "movss %1, %0"
275             : "=m" (*f3), "+x" (*f1)
276             : "x" (*f2));
277 }

279 extern __inline__ void
280 sse_addss(float *f1, float *f2, float *f3)
281 {
282         __asm__ __volatile__(
283             "addss %2, %1\n\t"
284             "movss %1, %0"
285             : "=m" (*f3), "+x" (*f1)
286             : "x" (*f2));
287 }

289 extern __inline__ void
290 sse_subss(float *f1, float *f2, float *f3)
291 {
292         __asm__ __volatile__(
293             "subss %2, %1\n\t"
294             "movss %1, %0"
295             : "=m" (*f3), "+x" (*f1)
296             : "x" (*f2));
297 }

299 extern __inline__ void
300 sse_mulss(float *f1, float *f2, float *f3)
301 {
302         __asm__ __volatile__(
303             "mulss %2, %1\n\t"
304             "movss %1, %0"
305             : "=m" (*f3), "+x" (*f1)
306             : "x" (*f2));
307 }

309 extern __inline__ void
310 sse_divss(float *f1, float *f2, float *f3)
311 {
312         __asm__ __volatile__(
313             "divss %2, %1\n\t"
314             "movss %1, %0"
315             : "=m" (*f3), "+x" (*f1)
316             : "x" (*f2));
```



```
317 }

319 extern __inline__ void
320 sse_sqrtss(float *f1, float *f2)
321 {
322         double tmp;

324         __asm__ __volatile__(
325             "sqrtss %2, %1\n\t"
326             "movss  %1, %0"
327             : "=m" (*f2), "=x" (tmp)
328             : "m" (*f1));
329 }

331 extern __inline__ void
332 sse_ucomiss(float *f1, float *f2)
333 {
334         __asm__ __volatile__("ucomiss %1, %0" : : "x" (*f1), "x" (*f2));

336 }

338 extern __inline__ void
339 sse_comiss(float *f1, float *f2)
340 {
341         __asm__ __volatile__("comiss %1, %0" : : "x" (*f1), "x" (*f2));
342 }

344 extern __inline__ void
345 sse_cvtss2sd(float *f1, double *d1)
346 {
347         double tmp;

349         __asm__ __volatile__(
350             "cvtss2sd %2, %1\n\t"
351             "movsd    %1, %0"
352             : "=m" (*d1), "=x" (tmp)
353             : "m" (*f1));
354 }

356 extern __inline__ void
357 sse_cvtsi2ss(int *i1, float *f1)
358 {
359         double tmp;

361         __asm__ __volatile__(
362             "cvtsi2ss %2, %1\n\t"
363             "movss    %1, %0"
364             : "=m" (*f1), "=x" (tmp)
365             : "m" (*i1));
366 }

368 extern __inline__ void
369 sse_cvttss2si(float *f1, int *i1)
370 {
371         int tmp;

373         __asm__ __volatile__(
374             "cvttss2si %2, %1\n\t"
375             "movl      %1, %0"
376             : "=m" (*i1), "=r" (tmp)
377             : "m" (*f1));
378 }

380 extern __inline__ void
381 sse_cvtss2si(float *f1, int *i1)
382 {
```

```
383         int tmp;
384
385         __asm__ __volatile__(
386             "cvtss2si %2, %1\n\t"
387             "movl     %1, %0"
388             : "=m" (*i1), "=r" (tmp)
389             : "m" (*f1));
390 }
391
392 #if defined(__amd64)
393 extern __inline__ void
394 sse_cvtsi2ssq(long long *ll1, float *f1)
395 {
396         double tmp;
397
398         __asm__ __volatile__(
399             "cvtsi2ssq %2, %1\n\t"
400             "movss     %1, %0"
401             : "=m" (*f1), "=x" (tmp)
402             : "m" (*ll1));
403 }
404
405 extern __inline__ void
406 sse_cvttss2siq(float *f1, long long *ll1)
407 {
408         uint64_t tmp;
409
410         __asm__ __volatile__(
411             "cvttss2siq %2, %1\n\t"
412             "movq       %1, %0"
413             : "=m" (*ll1), "=r" (tmp)
414             : "m" (*f1));
415 }
416
417 extern __inline__ void
418 sse_cvtss2siq(float *f1, long long *ll1)
419 {
420         uint64_t tmp;
421
422         __asm__ __volatile__(
423             "cvtss2siq %2, %1\n\t"
424             "movq      %1, %0"
425             : "=m" (*ll1), "=r" (tmp)
426             : "m" (*f1));
427 }
428
429 #endif
430
431 extern __inline__ void
432 sse_cmpeqsd(double *d1, double *d2, long long *ll1)
433 {
434         __asm__ __volatile__(
435             "cmpeqsd %2,%1\n\t"
436             "movsd   %1,%0"
437             : "=m" (*ll1), "+x" (*d1)
438             : "x" (*d2));
439 }
440
441 extern __inline__ void
442 sse_cmpltsd(double *d1, double *d2, long long *ll1)
443 {
444         __asm__ __volatile__(
445             "cmpltsd %2,%1\n\t"
446             "movsd   %1,%0"
447             : "=m" (*ll1), "+x" (*d1)
448             : "x" (*d2));
```



```
449 }
450
451 extern __inline__ void
452 sse_cmplesd(double *d1, double *d2, long long *ll1)
453 {
454         __asm__ __volatile__(
455             "cmplesd %2,%1\n\t"
456             "movsd   %1,%0"
457             : "=m" (*ll1), "+x" (*d1)
458             : "x" (*d2));
459 }
460
461 extern __inline__ void
462 sse_cmpunordsd(double *d1, double *d2, long long *ll1)
463 {
464         __asm__ __volatile__(
465             "cmpunordsd %2,%1\n\t"
466             "movsd      %1,%0"
467             : "=m" (*ll1), "+x" (*d1)
468             : "x" (*d2));
469 }
470
471
472 extern __inline__ void
473 sse_minsd(double *d1, double *d2, double *d3)
474 {
475         __asm__ __volatile__(
476             "minsd %2,%1\n\t"
477             "movsd %1,%0"
478             : "=m" (*d3), "+x" (*d1)
479             : "x" (*d2));
480 }
481
482 extern __inline__ void
483 sse_maxsd(double *d1, double *d2, double *d3)
484 {
485         __asm__ __volatile__(
486             "maxsd %2,%1\n\t"
487             "movsd %1,%0"
488             : "=m" (*d3), "+x" (*d1)
489             : "x" (*d2));
490 }
491
492 extern __inline__ void
493 sse_addsd(double *d1, double *d2, double *d3)
494 {
495         __asm__ __volatile__(
496             "addsd %2,%1\n\t"
497             "movsd %1,%0"
498             : "=m" (*d3), "+x" (*d1)
499             : "x" (*d2));
500 }
501
502 extern __inline__ void
503 sse_subsd(double *d1, double *d2, double *d3)
504 {
505         __asm__ __volatile__(
506             "subsd %2,%1\n\t"
507             "movsd %1,%0"
508             : "=m" (*d3), "+x" (*d1)
509             : "x" (*d2));
510 }
511
512 extern __inline__ void
513 sse_mulsd(double *d1, double *d2, double *d3)
514 {
```

```
515         __asm__ __volatile__(
516             "mulsd %2,%1\n\t"
517             "movsd %1,%0"
518             : "=m" (*d3), "+x" (*d1)
519             : "x" (*d2));
520 }

522 extern __inline__ void
523 sse_divsd(double *d1, double *d2, double *d3)
524 {
525         __asm__ __volatile__(
526             "divsd %2,%1\n\t"
527             "movsd %1,%0"
528             : "=m" (*d3), "+x" (*d1)
529             : "x" (*d2));
530 }

532 extern __inline__ void
533 sse_sqrtsd(double *d1, double *d2)
534 {
535         double tmp;

537         __asm__ __volatile__(
538             "sqrtsd %2, %1\n\t"
539             "movsd %1, %0"
540             : "=m" (*d2), "=x" (tmp)
541             : "m" (*d1));
542 }

544 extern __inline__ void
545 sse_ucomisd(double *d1, double *d2)
546 {
547         __asm__ __volatile__("ucomisd %1, %0" : : "x" (*d1), "x" (*d2));
548 }

550 extern __inline__ void
551 sse_comisd(double *d1, double *d2)
552 {
553         __asm__ __volatile__("comisd %1, %0" : : "x" (*d1), "x" (*d2));
554 }

556 extern __inline__ void
557 sse_cvtsd2ss(double *d1, float *f1)
558 {
559         double tmp;

561         __asm__ __volatile__(
562             "cvtsd2ss %2,%1\n\t"
563             "movss    %1,%0"
564             : "=m" (*f1), "=x" (tmp)
565             : "m" (*d1));
566 }

568 extern __inline__ void
569 sse_cvtsi2sd(int *i1, double *d1)
570 {
571         double tmp;
572         __asm__ __volatile__(
573             "cvtsi2sd %2,%1\n\t"
574             "movsd    %1,%0"
575             : "=m" (*d1), "=x" (tmp)
576             : "m" (*i1));
577 }

579 extern __inline__ void
580 sse_cvttsd2si(double *d1, int *i1)
```



```
581 {
582         int tmp;

584         __asm__ __volatile__(
585             "cvttsd2si %2,%1\n\t"
586             "movl      %1,%0"
587             : "=m" (*i1), "=r" (tmp)
588             : "m" (*d1));
589 }

591 extern __inline__ void
592 sse_cvtsd2si(double *d1, int *i1)
593 {
594         int tmp;

596         __asm__ __volatile__(
597             "cvtsd2si %2,%1\n\t"
598             "movl     %1,%0"
599             : "=m" (*i1), "=r" (tmp)
600             : "m" (*d1));
601 }

603 #if defined(__amd64)
604 extern __inline__ void
605 sse_cvtsi2sdq(long long *ll1, double *d1)
606 {
607         double tmp;

609         __asm__ __volatile__(
610             "cvtsi2sdq %2,%1\n\t"
611             "movsd     %1,%0"
612             : "=m" (*d1), "=x" (tmp)
613             : "m" (*ll1));
614 }

616 extern __inline__ void
617 sse_cvttsd2siq(double *d1, long long *ll1)
618 {
619         uint64_t tmp;

621         __asm__ __volatile__(
622             "cvttsd2siq %2,%1\n\t"
623             "movq       %1,%0"
624             : "=m" (*ll1), "=r" (tmp)
625             : "m" (*d1));
626 }

628 extern __inline__ void
629 sse_cvtsd2siq(double *d1, long long *ll1)
630 {
631         uint64_t tmp;

633         __asm__ __volatile__(
634             "cvtsd2siq %2,%1\n\t"
635             "movq      %1,%0"
636             : "=m" (*ll1), "=r" (tmp)
637             : "m" (*d1));
638 }
639 #endif

641 #elif defined(__sparc)
642 extern __inline__ void
643 __fenv_getfsr(unsigned long *l)
644 {
645         __asm__ __volatile__(
646 #if defined(__sparcv9)
```

```
647                 "stx %%fsr,%0\n\t"
648 #else
649                 "st  %%fsr,%0\n\t"
650 #endif
651                 : "=m" (*l));
652 }

654 extern __inline__ void
655 __fenv_setfsr(const unsigned long *l)
656 {
657         __asm__ __volatile__(
658 #if defined(__sparcv9)
659                 "ldx %0,%%fsr\n\t"
660 #else
661                 "ld %0,%%fsr\n\t"
662 #endif
663                 : : "m" (*l) : "cc");
664 }

666 extern __inline__ void
667 __fenv_getfsr32(unsigned int *l)
668 {
669         __asm__ __volatile__("st %%fsr,%0\n\t" : "=m" (*l));
670 }

672 extern __inline__ void
673 __fenv_setfsr32(const unsigned int *l)
674 {
675         __asm__ __volatile__("ld %0,%%fsr\n\t" : : "m" (*l));
676 }
677 #else
678 #error "GCC FENV inlines not implemented for this platform"
679 #endif

681 #ifdef __cplusplus
682 }
683 #endif

685 #endif  /* __GNUC__ */

687 #endif /* _FENV_INLINES_H */
```

```
**********************************************************
    3216 Thu Oct  9 19:48:54 2014
new/usr/src/lib/libm/common/m9x/nearbyint.c
remove unused code from common/m9x/nearbyint.c
minor changes
patch07 - removed dead code with mtsk.h
patch06 - libm: fixed compilation issues after updates
patch05 - fixed amd64 issues with LIBM
patch01 - 693 import Sun Devpro Math Library
remove unused code from common/m9x/nearbyint.c
minor changes
patch07 - removed dead code with mtsk.h
patch06 - libm: fixed compilation issues after updates
patch05 - fixed amd64 issues with LIBM
patch01 - 693 import Sun Devpro Math Library
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" 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.
 */

#if defined(ELFOBJ)
#pragma weak nearbyint = __nearbyint
#endif

/*
 * 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_synonyms.h"
#include <fenv.h>

double
__nearbyint(double x) {
        union {
                unsigned i[2];
```



```
                double d;
        } xx;
        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);
                        /* assumes sparc-like QNaN */
#else
                        return (x + x);
#endif
                return (x);
        } else if ((hx | xx.i[LOWORD]) == 0)    /* x is zero */
                return (x);

        /* 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])))
                        xx.i[HIWORD] = sx | 0x3ff00000;
                else
                        xx.i[HIWORD] = sx;
                xx.i[LOWORD] = 0;
                return (xx.d);
        }

        /* round x at the integer bit */
        j = 0x433 - (hx >> 20);
        if (j >= 32) {
                i = 1 << (j - 32);
                frac = ((xx.i[HIWORD] << 1) << (63 - j)) |
                        (xx.i[LOWORD] >> (j - 32));
                if (xx.i[LOWORD] & (i - 1))
                        frac |= 1;
                if (!frac)
                        return (x);
                xx.i[LOWORD] = 0;
                xx.i[HIWORD] &= ~(i - 1);
                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);
                if (!frac)
                        return (x);
                xx.i[LOWORD] &= ~(i - 1);
                if ((rm == FE_UPWARD) || ((rm == FE_TONEAREST) &&
                        (frac > 0x80000000u || ((frac == 0x80000000) &&
                        (xx.i[LOWORD] & i)))) {
                        xx.i[LOWORD] += i;
```

```
117                         if (xx.i[LOWORD] == 0)
118                                 xx.i[HIWORD]++;
119                 }
120         }
121         return (xx.d);
122 }
```

```
**********************************************************
    2431 Thu Oct  9 19:48:54 2014
new/usr/src/lib/libm/common/m9x/scalblnl.c
remove -Wno-switch -Wno-parentheses -Wno-unused-variable from libm
rollback ISINFNANL in libm/common/m9x/scalblnl.c
patch07 - removed dead code with mtsk.h
patch06 - libm: fixed compilation issues after updates
patch01 - 693 import Sun Devpro Math Library
remove -Wno-switch -Wno-parentheses -Wno-unused-variable from libm
rollback ISINFNANL in libm/common/m9x/scalblnl.c
patch07 - removed dead code with mtsk.h
patch06 - libm: fixed compilation issues after updates
patch01 - 693 import Sun Devpro Math Library
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" 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.
 */

#if defined(ELFOBJ)
#pragma weak scalblnl = __scalblnl
#endif

#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 ISINFNANL(k, x) (k == 0x7fff)
#define XTWOT_OFFSET    113
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
#if defined(HANDLE_UNSUPPORTED)
#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
static const long double xtwot = 18446744073709551616.0L,       /* 2^64 */
        twomtm1 = 2.7105054312137610850186E-20L;                /* 2^-65 */
#endif

long double
scalblnl(long double x, long n) {
        int k = XBIASED_EXP(x);

        if (ISINFNANL(k, x))
                return (x + x);
        if (ISZEROL(x) || n == 0)
                return (x);
        if (k == 0) {
                x *= xtwot;
                k = XBIASED_EXP(x) - XTWOT_OFFSET;
        }
        k += (int) n;
        if (n > 50000 || k > 0x7ffe)
                return (LDBL_MAX * copysignl(LDBL_MAX, x));
        if (n < -50000 || k <= -XTWOT_OFFSET - 1)
                return (LDBL_MIN * copysignl(LDBL_MIN, x));
        if (k > 0) {
                XSET_EXP(k, x);
                return (x);
        }
        k += XTWOT_OFFSET + 1;
        XSET_EXP(k, x);
        return (x * twomtm1);
}
```

```
**********************************************************
    7012 Thu Oct  9 19:48:55 2014
new/usr/src/lib/libm/i386/src/libm_inlines.h
libm/i386/src/libm_inlines.h - ceil()
libm/i386/src/libm_inlines.h use xorl %0, %0 to get a 0
i386/src/libm_inlines.h isnan() was missing
libm fixes from richlowe - richlowe.net/webrevs/il_keith
patch01 - 693 import Sun Devpro Math Library
libm/i386/src/libm_inlines.h - ceil()
libm/i386/src/libm_inlines.h use xorl %0, %0 to get a 0
i386/src/libm_inlines.h isnan() was missing
libm fixes from richlowe - richlowe.net/webrevs/il_keith
patch01 - 693 import Sun Devpro Math Library
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL 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 __inline__ double
__inline_sqrt(double a)
{
        double ret;
```



```
        __asm__ __volatile__("fsqrt\n\t" : "=t" (ret) : "0" (a) : "cc");
        return (ret);
}

extern __inline__ double
__ieee754_sqrt(double a)
{
        return (__inline_sqrt(a));
}

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

        __asm__ __volatile__("fsqrt\n\t" : "=t" (ret) : "0" (a) : "cc");
        return (ret);
}

extern __inline__ double
__inline_rint(double a)
{
        __asm__ __volatile__(
            "andl $0x7fffffff,%1\n\t"
            "cmpl $0x43300000,%1\n\t"
            "jae  1f\n\t"
            "frndint\n\t"
            "1: fwait\n\t"
            : "+t" (a), "+&r" (_HI_WORD(a))
            :
            : "cc");

        return (a);
}

/*
 * 00 - 24 bits
 * 01 - reserved
 * 10 - 53 bits
 * 11 - 64 bits
 */
extern __inline__ int
__swapRP(int i)
{
        int ret;
        uint16_t cw;

        __asm__ __volatile__("fstcw %0\n\t" : "=m" (cw));

        ret = (cw >> 8) & 0x3;
        cw = (cw & 0xfcff) | ((i & 0x3) << 8);

        __asm__ __volatile__("fldcw %0\n\t" : : "m" (cw));

        return (ret);
}

/*
 * 00 - Round to nearest, with even preferred
 * 01 - Round down
 * 10 - Round up
 * 11 - Chop
 */
extern __inline__ enum fp_direction_type
__swap87RD(enum fp_direction_type i)
```

```
{
        int ret;
        uint16_t cw;

        __asm__ __volatile__("fstcw %0\n\t" : "=m" (cw));

        ret = (cw >> 10) & 0x3;
        cw = (cw & 0xf3ff) | ((i & 0x3) << 10);

        __asm__ __volatile__("fldcw %0\n\t" : : "m" (cw));

        return (ret);
}

extern __inline__ double
ceil(double d)
{
        /*
         * Let's set a Rounding Control (RC) bits from x87 FPU Control Word
         * to fp_positive and save old bits in rd.
         */
        short rd = __swap87RD(fp_positive);

        /*
         * The FRNDINT instruction returns a floating-point value that is the
         * integral value closest to the source value in the direction of the
         * rounding mode specified in the RC field of the x87 FPU control word.
         *
         * Rounds the source value in the ST(0) register to the nearest
         * integral value, depending on the current rounding mode
         * (setting of the RC field of the FPU control word),
         * and stores the result in ST(0).
         */
        __asm__ __volatile__("frndint" : "+t" (d) : : "cc");

        /* restore old RC bits */
        __swap87RD(rd);

        return (d);
}

extern __inline__ double
copysign(double d1, double d2)
{
        __asm__ __volatile__(
            "andl $0x7fffffff,%0\n\t"   /* %0 <-- hi_32(abs(d)) */
            "andl $0x80000000,%1\n\t"   /* %1[31] <-- sign_bit(d2) */
            "orl  %1,%0\n\t"            /* %0 <-- hi_32(copysign(x,y)) */
            : "+&r" (_HI_WORD(d1)), "+r" (_HI_WORD(d2))
            :
            : "cc");

        return (d1);
}

extern __inline__ double
fabs(double d)
{
        __asm__ __volatile__("fabs\n\t" : "+t" (d) : : "cc");
        return (d);
}

extern __inline__ float
fabsf(float d)
{
        __asm__ __volatile__("fabs\n\t" : "+t" (d) : : "cc");
```



```
        return (d);
}

extern __inline__ long double
fabsl(long double d)
{
        __asm__ __volatile__("fabs\n\t" : "+t" (d) : : "cc");
        return (d);
}

extern __inline__ int
finite(double d)
{
        int ret = _HI_WORD(d);

        __asm__ __volatile__(
            "notl %0\n\t"
            "andl $0x7ff00000,%0\n\t"
            "negl %0\n\t"
            "shrl $31,%0\n\t"
            : "+r" (ret)
            :
            : "cc");
        return (ret);
}

extern __inline__ double
floor(double d)
{
        short rd = __swap87RD(fp_negative);

        __asm__ __volatile__("frndint" : "+t" (d), "+r" (rd) : : "cc");
        __swap87RD(rd);

        return (d);
}

/*
 *      branchless __isnan
 *      ((0x7ff00000-[((lx|-lx)>>31)&1]|ahx)>>31)&1 = 1 iff x is NaN
 */
extern __inline__ int
isnan(double d)
{
        int ret;

        __asm__ __volatile__(
                "movl %1,%%ecx\n\t"
                "negl %%ecx\n\t"                        /* ecx <-- -lo_32(x) */
                "orl  %%ecx,%1\n\t"
                "shrl $31,%1\n\t"                       /* 1 iff lx != 0 */
                "andl $0x7fffffff,%2\n\t"       /* ecx <-- hi_32(abs(x)) */
                "orl  %2,%1\n\t"
                "subl $0x7ff00000,%1\n\t"
                "negl %1\n\t"
                "shrl $31,%1\n\t"
                : "=r" (ret)
                : "0" (_HI_WORD(d)), "r" (_LO_WORD(d))
                : "ecx");

        return (ret);
}

extern __inline__ int
isnanf(float f)
{
```

```
251         __asm__ __volatile__(
252             "andl $0x7fffffff,%0\n\t"
253             "negl %0\n\t"
254             "addl $0x7f800000,%0\n\t"
255             "shrl $31,%0\n\t"
256             : "+r" (f)
257             :
258             : "cc");

260         return (f);
261 }

263 extern __inline__ double
264 rint(double a) {
265     return (__inline_rint(a));
266 }

268 extern __inline__ double
269 scalbn(double d, int n)
270 {
271         double dummy;

273         __asm__ __volatile__(
274             "fildl %2\n\t"      /* Convert N to extended */
275             "fxch\n\t"
276             "fscale\n\t"
277             : "+t" (d), "=u" (dummy)
278             : "m" (n)
279             : "cc");

281         return (d);
282 }

284 extern __inline__ int
285 signbit(double d)
286 {
287         return (_HI_WORD(d) >> 31);
288 }

290 extern __inline__ int
291 signbitf(float f)
292 {
293         return ((*(uint32_t *)&f) >> 31);
294 }

296 extern __inline__ double
297 sqrt(double d)
298 {
299         return (__inline_sqrt(d));
300 }

302 extern __inline__ float
303 sqrtf(float f)
304 {
305         return (__inline_sqrtf(f));
306 }

308 extern __inline__ long double
309 sqrtl(long double ld)
310 {
311         __asm__ __volatile__("fsqrt" : "+t" (ld) : : "cc");
312         return (ld);
313 }

315 extern __inline__ int
316 isnanl(long double ld)
```



```
317 {
318         int ret = _HIER_WORD(ld);

320         __asm__ __volatile__(
321             "andl  $0x00007fff,%0\n\t"
322             "jz    1f\n\t"              /* jump if exp is all 0 */
323             "xorl  $0x00007fff,%0\n\t"
324             "jz    2f\n\t"              /* jump if exp is all 1 */
325             "testl $0x80000000,%1\n\t"
326             "jz    3f\n\t"              /* jump if leading bit is 0 */
327             "xorl  %0,%0\n\t"
328             "jmp   1f\n\t"
329             "2:\n\t"                    /* note that %0 = 0 from before */
330             "cmpl  $0x80000000,%1\n\t"  /* what is first half of significand? */
331             "jnz   3f\n\t"              /* jump if not equal to 0x80000000 */
332             "testl $0xffffffff,%2\n\t"  /* is second half of significand 0? */
333             "jnz   3f\n\t"              /* jump if not equal to 0 */
334             "jmp   1f\n\t"
335             "3:\n\t"
336             "movl  $1,%0\n\t"
337             "1:\n\t"
338             : "+&r" (ret)
339             : "r" (_HI_WORD(ld)), "r" (_LO_WORD(ld))
340             : "cc");

342         return (ret);
343 }

345 #ifdef __cplusplus
346 }
347 #endif

349 #endif  /* __GNUC__ */

351 #endif /* _LIBM_INLINES_H */
```

```
**********************************************************
    7033 Thu Oct  9 19:48:55 2014
new/usr/src/lib/libm/sparc/src/libm_inlines.h
libm/sparc/src/libm_inlines.h - __swapTE()
libm/sparc/src/libm_inlines.h - __swapEX()
libm/sparc/src/libm_inlines.h - fixes part 1
libm fixes from richlowe - richlowe.net/webrevs/il_keith
patch01 - 693 import Sun Devpro Math Library
libm/sparc/src/libm_inlines.h - fixes part 1
libm fixes from richlowe - richlowe.net/webrevs/il_keith
patch01 - 693 import Sun Devpro Math Library
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL 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 __inline__ double
__inline_sqrt(double d)
{
        double ret;

        __asm__ __volatile__("fsqrtd %1,%0\n\t" : "=e" (ret) : "e" (d));
        return (ret);
}

extern __inline__ float
```



```
__inline_sqrtf(float f)
{
        float ret;

        __asm__ __volatile__("fsqrts %1,%0\n\t" : "=f" (ret) : "f" (f));
        return (ret);
}

extern __inline__ enum fp_class_type
fp_classf(float f)
{
        enum fp_class_type ret;
        uint32_t tmp;

        /* XXX: Separate input and output */
        __asm__ __volatile__(
            "sethi  %%hi(0x80000000),%1\n\t"
            "andncc %2,%1,%0\n\t"
            "bne    1f\n\t"
            "nop\n\t"
            "mov    0,%0\n\t"
            "ba 2f\n\t"         /* x is 0 */
            "nop\n\t"
            "1:\n\t"
            "sethi  %%hi(0x7f800000),%1\n\t"
            "andcc  %0,%1,%%g0\n\t"
            "bne    1f\n\t"
            "nop\n\t"
            "mov    1,%0\n\t"
            "ba     2f\n\t"     /* x is subnormal */
            "nop\n\t"
            "1:\n\t"
            "cmp    %0,%1\n\t"
            "bge    1f\n\t"
            "nop\n\t"
            "mov    2,%0\n\t"
            "ba     2f\n\t"     /* x is normal */
            "nop\n\t"
            "1:\n\t"
            "bg     1f\n\t"
            "nop\n\t"
            "mov    3,%0\n\t"
            "ba     2f\n\t"     /* x is __infinity */
            "nop\n\t"
            "1:\n\t"
            "sethi  %%hi(0x00400000),%1\n\t"
            "andcc  %0,%1,%%g0\n\t"
            "mov    4,%0\n\t"   /* x is quiet NaN */
            "bne    2f\n\t"
            "nop\n\t"
            "mov    5,%0\n\t"   /* x is signaling NaN */
            "2:\n\t"
            : "=r" (ret), "=&r" (tmp)
            : "r" (f)
            : "cc");
        return (ret);
}

#define _HI_WORD(x)     ((uint32_t *)&x)[0]
#define _LO_WORD(x)     ((uint32_t *)&x)[1]

extern __inline__ enum fp_class_type
fp_class(double d)
{
        enum fp_class_type ret;
        uint32_t tmp;
```

```
122         __asm__ __volatile__(
123             "sethi %%hi(0x80000000),%1\n\t"     /* %1 gets 80000000 */
124             "andn  %2,%1,%0\n\t"                /* %2-%0 gets abs(x) */
125             "orcc  %0,%3,%%g0\n\t"              /* set cc as x is zero/nonzero *
126             "bne   1f\n\t"                      /* branch if x is nonzero */
127             "nop\n\t"
128             "mov   0,%0\n\t"
129             "ba    2f\n\t"                      /* x is 0 */
130             "nop\n\t"
131             "1:\n\t"
132             "sethi %%hi(0x7ff00000),%1\n\t"     /* %1 gets 7ff00000 */
133             "andcc %0,%1,%%g0\n\t"              /* cc set by __exp field of x */
134             "bne   1f\n\t"                      /* branch if normal or max __exp
135             "nop\n\t"
136             "mov   1,%0\n\t"
137             "ba    2f\n\t"                      /* x is subnormal */
138             "nop\n\t"
139             "1:\n\t"
140             "cmp   %0,%1\n\t"
141             "bge   1f\n\t"                      /* branch if x is max __exp */
142             "nop\n\t"
143             "mov   2,%0\n\t"
144             "ba    2f\n\t"                      /* x is normal */
145             "nop\n\t"
146             "1:\n\t"
147             "andn  %0,%1,%0\n\t"                /* o0 gets msw __significand fie
148             "orcc  %0,%3,%%g0\n\t"              /* set cc by OR __significand */
149             "bne   1f\n\t"                      /* Branch if __nan */
150             "nop\n\t"
151             "mov   3,%0\n\t"
152             "ba    2f\n\t"                      /* x is __infinity */
153             "nop\n\t"
154             "1:\n\t"
155             "sethi %%hi(0x00080000),%1\n\t"
156             "andcc %0,%1,%%g0\n\t"              /* set cc by quiet/sig bit */
157             "be    1f\n\t"                      /* Branch if signaling */
158             "nop\n\t"
159             "mov   4,%0\n\t"                    /* x is quiet NaN */
160             "ba    2f\n\t"
161             "nop\n\t"
162             "1:\n\t"
163             "mov   5,%0\n\t"                    /* x is signaling NaN */
164             "2:\n\t"
165             : "=&r" (ret), "=&r" (tmp)
166             : "r" (_HI_WORD(d)), "r" (_LO_WORD(d))
167             : "cc");

169         return (ret);
170 }

172 extern __inline__ int
173 __swapEX(int i)
174 {
175         int ret;
176         uint32_t fsr;
177         uint32_t tmp1, tmp2;

179         __asm__ __volatile__(
180             "and  %4,0x1f,%2\n\t" /* tmp1 = %2 = %o1 */
181             "sll  %2,5,%2\n\t"  /* shift input to aexc bit location */
182             ".volatile\n\t"
183             "st   %%fsr,%1\n\t"
184             "ld   %1,%0\n\t"    /* %0 = fsr */
185             "andn %0,0x3e0,%3\n\t" /* tmp2 = %3 = %o2 */
186             "or   %2,%3,%2\n\t" /* %2 = new fsr */
```



```
187             "st   %2,%1\n\t"
188             "ld   %1,%%fsr\n\t"
189             "srl  %0,5,%0\n\t"
190             "and  %0,0x1f,%0\n\t" /* %0 = ret = %o0 */
191             ".nonvolatile\n\t"
192             : "=r" (ret), "=m" (fsr), "=r" (tmp1), "=r" (tmp2)
193             : "r" (i)
194             : "cc");

196         return (ret);
197 }

199 /*
200  * On the SPARC, __swapRP is a no-op; always return 0 for backward
201  * compatibility
202  */
203 /* ARGSUSED */
204 extern __inline__ enum fp_precision_type
205 __swapRP(enum fp_precision_type i)
206 {
207         return (0);
208 }

210 extern __inline__ enum fp_direction_type
211 __swapRD(enum fp_direction_type d)
212 {
213         enum fp_direction_type ret;
214         uint32_t fsr;
215         uint32_t tmp1, tmp2, tmp3;

217         __asm__ __volatile__(
218             "and  %5,0x3,%0\n\t"
219             "sll  %0,30,%2\n\t"         /* shift input to RD bit location */
220             ".volatile\n\t"
221             "st   %%fsr,%1\n\t"
222             "ld   %1,%0\n\t"            /* %0 = fsr */
223             "set  0xc0000000,%4\n\t"    /* mask of rounding direction bits */
224             "andn %0,%4,%3\n\t"
225             "or   %2,%3,%2\n\t"         /* %2 = new fsr */
226             "st   %2,%1\n\t"
227             "ld   %1,%%fsr\n\t"
228             "srl  %0,30,%0\n\t"
229             "and  %0,0x3,%0\n\t"
230             ".nonvolatile\n\t"
231             : "=r" (ret), "=m" (fsr), "=r" (tmp1), "=r" (tmp2), "=r" (tmp3)
232             : "r" (d)
233             : "cc");

235         return (ret);
236 }

238 extern __inline__ int
239 __swapTE(int i)
240 {
241         int ret;
242         uint32_t fsr, tmp1, tmp2;

244         __asm__ __volatile__(
245             "and  %4,0x1f,%0\n\t"
246             "sll  %0,23,%2\n\t"         /* shift input to TEM bit location */
247             ".volatile\n\t"
248             "st   %%fsr,%1\n\t"
249             "ld   %1,%0\n\t"            /* %0 = fsr */
250             "set  0x0f800000,%3\n\t"    /* mask of TEM (Trap Enable Mode bits) *
251             "andn %0,%3,%3\n\t"
252             "or   %2,%3,%2\n\t"         /* %2 = new fsr */
```

```
253             "st   %2,%1\n\t"
254             "ld   %1,%%fsr\n\t"
255             "srl  %0,23,%0\n\t"
256             "and  %0,0x1f,%0\n\t"
257             ".nonvolatile\n\t"
258             : "=r" (ret), "=m" (fsr), "=r" (tmp1), "=r" (tmp2)
259             : "r" (i)
260             : "cc");

262         return (ret);
263 }

265 extern __inline__ double
266 sqrt(double d)
267 {
268         return (__inline_sqrt(d));
269 }

271 extern __inline__ float
272 sqrtf(float f)
273 {
274         return (__inline_sqrtf(f));
275 }

277 extern __inline__ double
278 fabs(double d)
279 {
280         double ret;

282         __asm__ __volatile__("fabsd %1,%0\n\t" : "=e" (ret) : "e" (d));
283         return (ret);
284 }

286 extern __inline__ float
287 fabsf(float f)
288 {
289         float ret;

291         __asm__ __volatile__("fabss %1,%0\n\t" : "=f" (ret) : "f" (f));
292         return (ret);
293 }

295 #ifdef __cplusplus
296 }
297 #endif

299 #endif  /* __GNUC */

301 #endif /* _LIBM_INLINES_H */
```

```
**********************************************************
    6475 Thu Oct  9 19:48:55 2014
new/usr/src/lib/libm/sparcv9/src/libm_inlines.h
libm/sparcv9/src/libm_inlines.h - fabss/fabsd register should be listed as read
libm fixes from richlowe - richlowe.net/webrevs/il_keith
patch01 - 693 import Sun Devpro Math Library
libm fixes from richlowe - richlowe.net/webrevs/il_keith
patch01 - 693 import Sun Devpro Math Library
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL 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 __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);
}

extern __inline__ enum fp_class_type
fp_class(double d)
{
        enum fp_class_type ret;
        uint64_t dint;          /* Scratch for d-as-long */
        uint64_t tmp;

        __asm__ __volatile__(
            "fabsd  %3,%3\n\t"
            "std    %3,%1\n\t"
            "ldx    %1,%0\n\t"
            "orcc   %%g0,%0,%%g0\n\t"
            "be,pn  %%xcc,2f\n\t"
            "nop\n\t"
            "sethi  %%hi(0x7ff00000),%2\n\t"
            "sllx   %2,32,%2\n\t"
            "andcc  %0,%2,%%g0\n\t"
            "bne,pt %%xcc,1f\n\t"
            "nop\n\t"
            "or     %%g0,1,%0\n\t"
            "ba     2f\n\t"
            "nop\n\t"
            "1:\n\t"
            "subcc  %0,%2,%%g0\n\t"
            "bge,pn %%xcc,1f\n\t"
            "nop\n\t"
            "or     %%g0,2,%0\n\t"
            "ba     2f\n\t"
            "nop\n\t"
```

```
124             "1:\n\t"
125             "andncc %0,%2,%0\n\t"
126             "bne,pn %%xcc,1f\n\t"
127             "nop\n\t"
128             "or     %%g0,3,%0\n\t"
129             "ba     2f\n\t"
130             "nop\n\t"
131             "1:\n\t"
132             "sethi  %%hi(0x00080000),%2\n\t"
133             "sllx   %2,32,%2\n\t"
134             "andcc  %0,%2,%%g0\n\t"
135             "or     %%g0,4,%0\n\t"
136             "bne,pt %%xcc,2f\n\t"
137             "nop\n\t"
138             "or     %%g0,5,%0\n\t"
139             "2:\n\t"
140             : "=r" (ret), "=m" (dint), "=r" (tmp), "+e" (d)
141             :
142             : "cc");
143
144         return (ret);
145 }
146
147 extern __inline__ float
148 __inline_sqrtf(float f)
149 {
150         float ret;
151
152         __asm__ __volatile__("fsqrts %1,%0\n\t" : "=f" (ret) : "f" (f));
153         return (ret);
154 }
155
156 extern __inline__ double
157 __inline_sqrt(double d)
158 {
159         double ret;
160
161         __asm__ __volatile__("fsqrtd %1,%0\n\t" : "=f" (ret) : "f" (d));
162         return (ret);
163 }
164
165 extern __inline__ int
166 __swapEX(int i)
167 {
168         int ret;
169         uint32_t fsr;
170         uint64_t tmp1, tmp2;
171
172         __asm__ __volatile__(
173             "and  %4,0x1f,%2\n\t"
174             "sll  %2,5,%2\n\t"  /* shift input to aexc bit location */
175             ".volatile\n\t"
176             "st   %%fsr,%1\n\t"
177             "ld   %1,%0\n\t"    /* %0 = fsr */
178             "andn %0,0x3e0,%3\n\t"
179             "or   %2,%3,%2\n\t" /* %2 = new fsr */
180             "st   %2,%1\n\t"
181             "ld   %1,%%fsr\n\t"
182             "srl  %0,5,%0\n\t"
183             "and  %0,0x1f,%0\n\t"
184             ".nonvolatile\n\t"
185             : "=r" (ret), "=m" (fsr), "=r" (tmp1), "=r" (tmp2)
186             : "r" (i)
187             : "cc");
188
189         return (ret);
```



```
190 }
191
192 /*
193  * On the SPARC, __swapRP is a no-op; always return 0 for backward
194  * compatibility
195  */
196 /* ARGSUSED */
197 extern __inline__ enum fp_precision_type
198 __swapRP(enum fp_precision_type i)
199 {
200         return (0);
201 }
202
203 extern __inline__ enum fp_direction_type
204 __swapRD(enum fp_direction_type d)
205 {
206         enum fp_direction_type ret;
207         uint32_t fsr;
208         uint64_t tmp1, tmp2, tmp3;
209
210         __asm__ __volatile__(
211             "and   %5,0x3,%0\n\t"
212             "sll   %0,30,%2\n\t"        /* shift input to RD bit location */
213             ".volatile\n\t"
214             "st    %%fsr,%1\n\t"
215             "ld    %1,%0\n\t"           /* %0 = fsr */
216             /* mask of rounding direction bits */
217             "sethi %%hi(0xc0000000),%4\n\t"
218             "andn  %0,%4,%3\n\t"
219             "or    %2,%3,%2\n\t"        /* %2 = new fsr */
220             "st    %2,%1\n\t"
221             "ld    %1,%%fsr\n\t"
222             "srl   %0,30,%0\n\t"
223             "and   %0,0x3,%0\n\t"
224             ".nonvolatile\n\t"
225             : "=r" (ret), "=m" (fsr), "=r" (tmp1), "=r" (tmp2), "=r" (tmp3)
226             : "r" (d)
227             : "cc");
228
229         return (ret);
230 }
231
232 extern __inline__ int
233 __swapTE(int i)
234 {
235         int ret;
236         uint32_t fsr;
237         uint64_t tmp1, tmp2, tmp3;
238
239         __asm__ __volatile__(
240             "and   %5,0x1f,%0\n\t"
241             "sll   %0,23,%2\n\t"        /* shift input to TEM bit location */
242             ".volatile\n\t"
243             "st    %%fsr,%1\n\t"
244             "ld    %1,%0\n\t"           /* %0 = fsr */
245             /* mask of TEM (Trap Enable Mode bits) */
246             "sethi %%hi(0x0f800000),%4\n\t"
247             "andn  %0,%4,%3\n\t"
248             "or    %2,%3,%2\n\t"        /* %2 = new fsr */
249             "st    %2,%1\n\t"
250             "ld    %1,%%fsr\n\t"
251             "srl   %0,23,%0\n\t"
252             "and   %0,0x1f,%0\n\t"
253             ".nonvolatile\n\t"
254             : "=r" (ret), "=m" (fsr), "=r" (tmp1), "=r" (tmp2), "=r" (tmp3)
255             : "r" (i)
```

```
256             : "cc");

258         return (ret);
259 }


262 extern __inline__ double
263 sqrt(double d)
264 {
265         return (__inline_sqrt(d));
266 }

268 extern __inline__ float
269 sqrtf(float f)
270 {
271         return (__inline_sqrtf(f));
272 }

274 extern __inline__ double
275 fabs(double d)
276 {
277         double ret;

279         __asm__ __volatile__("fabsd %1,%0\n\t" : "=e" (ret) : "e" (d));
280         return (ret);
281 }

283 extern __inline__ float
284 fabsf(float f)
285 {
286         float ret;

288         __asm__ __volatile__("fabss %1,%0\n\t" : "=f" (ret) : "f" (f));
289         return (ret);
290 }

292 #ifdef __cplusplus
293 }
294 #endif

296 #endif  /* __GNUC__ */

298 #endif /* _LIBM_INLINES_H */
```

```
**********************************************************
    5733 Thu Oct  9 19:48:55 2014
new/usr/src/lib/libmvec/Makefile.com
libmvec - remove -Wno-parentheses and -Wno-unused-variable
comment in tgamma*.c
libm/common/m9x/tgamma.c
remove -Wno-uninitialized in libmvec
libm fixes from richlowe - richlowe.net/webrevs/il_keith
patch01 - 693 import Sun Devpro Math Library
comment in tgamma*.c
libm/common/m9x/tgamma.c
remove -Wno-uninitialized in libmvec
libm fixes from richlowe - richlowe.net/webrevs/il_keith
patch01 - 693 import Sun Devpro Math Library
**********************************************************
#
# 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 Nexenta Systems, Inc.  All rights reserved.
#

LIBMDIR         = $(SRC)/lib/libm

mvecOBJS        = \
                  __vTBL_atan1.o \
                  __vTBL_atan2.o \
                  __vTBL_rsqrt.o \
                  __vTBL_sincos.o \
                  __vTBL_sincos2.o \
                  __vTBL_sqrtf.o \
                  __vatan.o \
                  __vatan2.o \
                  __vatan2f.o \
                  __vatanf.o \
                  __vc_abs.o \
                  __vc_exp.o \
                  __vc_log.o \
                  __vc_pow.o \
                  __vcos.o \
                  __vcosbig.o \
                  __vcosbigf.o \
                  __vcosf.o \
                  __vexp.o \
                  __vexpf.o \
                  __vhypot.o \
                  __vhypotf.o \
                  __vlog.o \
                  __vlogf.o \
                  __vpow.o \
                  __vpowf.o \
                  __vrem_pio2m.o \
                  __vrhypot.o \
                  __vrhypotf.o \
                  __vrsqrt.o \
                  __vrsqrtf.o \
                  __vsin.o \
                  __vsinbig.o \
```



```
                  __vsinbigf.o \
                  __vsincos.o \
                  __vsincosbig.o \
                  __vsincosbigf.o \
                  __vsincosf.o \
                  __vsinf.o \
                  __vsqrt.o \
                  __vsqrtf.o \
                  __vz_abs.o \
                  __vz_exp.o \
                  __vz_log.o \
                  __vz_pow.o \
                  vatan2_.o \
                  vatan2f_.o \
                  vatan_.o \
                  vatanf_.o \
                  vc_abs_.o \
                  vc_exp_.o \
                  vc_log_.o \
                  vc_pow_.o \
                  vcos_.o \
                  vcosf_.o \
                  vexp_.o \
                  vexpf_.o \
                  vhypot_.o \
                  vhypotf_.o \
                  vlog_.o \
                  vlogf_.o \
                  vpow_.o \
                  vpowf_.o \
                  vrhypot_.o \
                  vrhypotf_.o \
                  vrsqrt_.o \
                  vrsqrtf_.o \
                  vsin_.o \
                  vsincos_.o \
                  vsincosf_.o \
                  vsinf_.o \
                  vsqrt_.o \
                  vsqrtf_.o \
                  vz_abs_.o \
                  vz_exp_.o \
                  vz_log_.o \
                  vz_pow_.o \
                  #end

mvecvisCOBJS    = \
                  __vTBL_atan1.o \
                  __vTBL_atan2.o \
                  __vTBL_rsqrt.o \
                  __vTBL_sincos.o \
                  __vTBL_sincos2.o \
                  __vTBL_sqrtf.o \
                  __vcosbig.o \
                  __vcosbigf.o \
                  __vrem_pio2m.o \
                  __vsinbig.o \
                  __vsinbigf.o \
                  __vsincosbig.o \
                  __vsincosbigf.o \
                  #end

mvecvisSOBJS    = \
                  __vatan.o \
                  __vatan2.o \
                  __vatan2f.o \
```

```
118                 __vatanf.o \
119                 __vcos.o \
120                 __vcosf.o \
121                 __vexp.o \
122                 __vexpf.o \
123                 __vhypot.o \
124                 __vhypotf.o \
125                 __vlog.o \
126                 __vlogf.o \
127                 __vpow.o \
128                 __vpowf.o \
129                 __vrhypot.o \
130                 __vrhypotf.o \
131                 __vrsqrt.o \
132                 __vrsqrtf.o \
133                 __vsin.o \
134                 __vsincos.o \
135                 __vsincosf.o \
136                 __vsinf.o \
137                 __vsqrt.o \
138                 __vsqrtf.o \
139                 #end

141 mvecvis2COBJS   = \
142                 __vTBL_sincos.o \
143                 __vTBL_sincos2.o \
144                 __vTBL_sqrtf.o \
145                 __vcosbig.o \
146                 __vcosbig_ultra3.o \
147                 __vrem_pio2m.o \
148                 __vsinbig.o \
149                 __vsinbig_ultra3.o \
150                 #end

152 mvecvis2SOBJS   = \
153                 __vcos_ultra3.o \
154                 __vlog_ultra3.o \
155                 __vsin_ultra3.o \
156                 __vsqrtf_ultra3.o \
157                 #end

159 include         $(SRC)/lib/Makefile.lib
160 include         $(SRC)/lib/Makefile.rootfs
161 include         $(LIBMDIR)/Makefile.libm.com

163 LIBS            = $(DYNLIB)
164 SRCDIR          = ../common/
165 DYNFLAGS        += -zignore

167 LINTERROFF      = -erroff=E_FP_DIVISION_BY_ZERO
168 LINTERROFF      += -erroff=E_FP_INVALID
169 LINTERROFF      += -erroff=E_BAD_PTR_CAST_ALIGN
170 LINTERROFF      += -erroff=E_ASSIGMENT_CAUSE_LOSS_PREC
171 LINTERROFF      += -erroff=E_FUNC_SET_NOT_USED

173 LINTFLAGS       += $(LINTERROFF)
174 LINTFLAGS64     += $(LINTERROFF)
175 LINTFLAGS64     += -errchk=longptr64

177 CLAGS           += $(LINTERROFF)
178 CFLAGS64        += $(LINTERROFF)

180 ASDEF           += -DLIBMVEC_SO_BUILD

182 FLTRPATH_sparc          = $$ORIGIN/cpu/$$ISALIST/libmvec_isa.so.1
183 FLTRPATH_sparcv9        = $$ORIGIN/../cpu/$$ISALIST/sparcv9/libmvec_isa.so.1
```



```
184 FLTRPATH_i386           = $$ORIGIN/libmvec/$$HWCAP
185 FLTRPATH                = $(FLTRPATH_$(TARGET_ARCH))

187 sparc_CFLAGS += -_cc=-W0,-xintrinsic
188 sparcv9_CFLAGS += -_cc=-W0,-xintrinsic
189 CPPFLAGS_i386   += -Dfabs=__fabs

191 CPPFLAGS        += -DLIBMVEC_SO_BUILD

193 SRCS_mvec_i386 = \
194         ../common/__vsqrtf.c \
195         #end

197 SRCS_mvec_sparc = \
198         $(SRCS_mvec_i386) \
199         #end
200 SRCS_mvec_sparcv9 = \
201         $(SRCS_mvec_i386) \
202         #end

204 SRCS_mvec = \
205         $(SRCS_mvec_$(TARGETMACH)) \
206         ../common/__vTBL_atan1.c \
207         ../common/__vTBL_atan2.c \
208         ../common/__vTBL_rsqrt.c \
209         ../common/__vTBL_sincos.c \
210         ../common/__vTBL_sincos2.c \
211         ../common/__vTBL_sqrtf.c \
212         ../common/__vatan.c \
213         ../common/__vatan2.c \
214         ../common/__vatan2f.c \
215         ../common/__vatanf.c \
216         ../common/__vc_abs.c \
217         ../common/__vc_exp.c \
218         ../common/__vc_log.c \
219         ../common/__vc_pow.c \
220         ../common/__vcos.c \
221         ../common/__vcosbig.c \
222         ../common/__vcosbigf.c \
223         ../common/__vcosf.c \
224         ../common/__vexp.c \
225         ../common/__vexpf.c \
226         ../common/__vhypot.c \
227         ../common/__vhypotf.c \
228         ../common/__vlog.c \
229         ../common/__vlogf.c \
230         ../common/__vpow.c \
231         ../common/__vpowf.c \
232         ../common/__vrem_pio2m.c \
233         ../common/__vrhypot.c \
234         ../common/__vrhypotf.c \
235         ../common/__vrsqrt.c \
236         ../common/__vrsqrtf.c \
237         ../common/__vsin.c \
238         ../common/__vsinbig.c \
239         ../common/__vsinbigf.c \
240         ../common/__vsincos.c \
241         ../common/__vsincosbig.c \
242         ../common/__vsincosbigf.c \
243         ../common/__vsincosf.c \
244         ../common/__vsinf.c \
245         ../common/__vsqrt.c \
246         ../common/__vz_abs.c \
247         ../common/__vz_exp.c \
248         ../common/__vz_log.c \
249         ../common/__vz_pow.c \
```

```
250         ../common/vatan2_.c \
251         ../common/vatan2f_.c \
252         ../common/vatan_.c \
253         ../common/vatanf_.c \
254         ../common/vc_abs_.c \
255         ../common/vc_exp_.c \
256         ../common/vc_log_.c \
257         ../common/vc_pow_.c \
258         ../common/vcos_.c \
259         ../common/vcosf_.c \
260         ../common/vexp_.c \
261         ../common/vexpf_.c \
262         ../common/vhypot_.c \
263         ../common/vhypotf_.c \
264         ../common/vlog_.c \
265         ../common/vlogf_.c \
266         ../common/vpow_.c \
267         ../common/vpowf_.c \
268         ../common/vrhypot_.c \
269         ../common/vrhypotf_.c \
270         ../common/vrsqrt_.c \
271         ../common/vrsqrtf_.c \
272         ../common/vsin_.c \
273         ../common/vsincos_.c \
274         ../common/vsincosf_.c \
275         ../common/vsinf_.c \
276         ../common/vsqrt_.c \
277         ../common/vsqrtf_.c \
278         ../common/vz_abs_.c \
279         ../common/vz_exp_.c \
280         ../common/vz_log_.c \
281         ../common/vz_pow_.c \
282         #end

284 .KEEP_STATE:

286 all:    $(LIBS)

288 lint:   lintcheck

290 pics/%.o: ../$(TARGET_ARCH)/src/%.S
291         $(COMPILE.s) -o $@ $<
292         $(POST_PROCESS_O)

294 pics/%.o: ../common/$$(CHIP)/%.S
295         $(COMPILE.s) -o $@ $<
296         $(POST_PROCESS_O)
```

```
**********************************************************
   29767 Thu Oct  9 19:48:55 2014
new/usr/src/lib/libmvec/common/__vcos.c
Revert "remove unused v from libmvec"
This reverts commit e853d278ee4b7f2a8c2117cc598cfc68b4e3f29b.
fix system-library-math.mf
update libm manifests
14071:dece9aafe99a - fix build problems on sparc
remove unused v from libmvec
fix for patch09 - use __GNU_UNUSED
patch12 - math.h: Align things with GCC
patch11 - added LIBM man pages
patch09 - update libmvec: fix build issues by gcc46
patch08 - libmvec: fixed compilation issues after updates
patch01 - 693 import Sun Devpro Math Library
Revert "remove unused v from libmvec"
This reverts commit e853d278ee4b7f2a8c2117cc598cfc68b4e3f29b.
fix system-library-math.mf
update libm manifests
14071:dece9aafe99a - fix build problems on sparc
remove unused v from libmvec
fix for patch09 - use __GNU_UNUSED
patch12 - math.h: Align things with GCC
patch11 - added LIBM man pages
patch09 - update libmvec: fix build issues by gcc46
patch08 - libmvec: fixed compilation issues after updates
patch01 - 693 import Sun Devpro Math Library
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" 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.
 */

#include <sys/isa_defs.h>
#include <sys/ccompile.h>

#ifdef _LITTLE_ENDIAN
#define HI(x)   *(1+(int*)x)
#define LO(x)   *(unsigned*)x
#else
#define HI(x)   *(int*)x
#define LO(x)   *(1+(unsigned*)x)
```



```
#endif

#ifdef __RESTRICT
#define restrict _Restrict
#else
#define restrict
#endif

/*
 * vcos.1.c
 *
 * Vector cosine function.  Just slight modifications to vsin.8.c, mainly
 * in the primary range part.
 *
 * Modification to primary range processing.  If an argument that does not
 * fall in the primary range is encountered, then processing is continued
 * in the medium range.
 *
 */

extern const double __vlibm_TBL_sincos_hi[], __vlibm_TBL_sincos_lo[];

static const double
        half[2] = { 0.5, -0.5 },
        one             = 1.0,
        invpio2 = 0.63661977236758134307553505,  /* 53 bits of pi/2 */
        pio2_1  = 1.57079632673412561416600,     /* first 33 bits of pi/2 */
        pio2_2  = 6.07710050630396597659600e-11, /* second 33 bits of pi/2 */
        pio2_3  = 2.02226248711166455796e-21, /* third 33 bits of pi/2 */
        pio2_3t = 8.47842766036889964395900e-32, /* pi/2 - pio2_3 */
        pp1             = -1.66666666660576046527626394313498255467600e-0001,
        pp2             =  8.33326120969096312671837656614618094444200e-0003,
        qq1             = -4.99999999977710986407023955908711557870000e-0001,
        qq2             =  4.16665486385721935064505588101884208958000e-0002,
        poly1[2]= { -1.66666666666629669805215138920301589656000e-0001,
                                -4.99999999999931701464060878888294524481000e-0001
        poly2[2]= {  8.33333333239095129568399345528033637666300e-0003,
                                 4.16666666639486191753564059396370822231900e-0002
        poly3[2]= { -1.98412623799797669279155177823009840396000e-0004,
                                -1.38888855265614286783275668773685168146200e-0003
        poly4[2]= {  2.75340362485427723764998762284833035111000e-0006,
                                 2.47851942368146079661812828945453052475900e-0005

static const unsigned thresh[2] = { 0x3fc90000, 0x3fc40000 };

/* Don't __ the following; acomp will handle it */
extern double fabs( double );
extern void __vlibm_vcos_big( int, double *, int, double *, int, int );

/*
 * y[i*stridey] := cos( x[i*stridex] ), for i = 0..n.
 *
 * Calls __vlibm_vcos_big to handle all elts which have abs >~ 1.647e+06.
 * Argument reduction is done here for elts pi/4 < arg < 1.647e+06.
 *
 * elts < 2^-27 use the approximation 1.0 ~ cos(x).
 */
void
__vcos( int n, double * restrict x, int stridex, double * restrict y,
        int stridey )
{
        double          x0_or_one[4], x1_or_one[4], x2_or_one[4];
        double          y0_or_zero[4], y1_or_zero[4], y2_or_zero[4];
        double          x0, x1, x2, *py0 = 0, *py1 = 0, *py2, *xsave, *ysave;
        unsigned        hx0, hx1, hx2, xsb0, xsb1 = 0, xsb2;
        int             i, biguns, nsave, sxsave, sysave;
```

```
105         volatile int    v __GNU_UNUSED;
106         nsave = n;
107         xsave = x;
108         sxsave = stridex;
109         ysave = y;
110         sysave = stridey;
111         biguns = 0;
112
113         do /* MAIN LOOP */
114         {
115                 /* Gotos here so _break_ exits MAIN LOOP. */
116 LOOP0:  /* Find first arg in right range. */
117                 xsb0 = HI(x); /* get most significant word */
118                 hx0 = xsb0 & ~0x80000000; /* mask off sign bit */
119                 if ( hx0 > 0x3fe921fb ) {
120                         /* Too big: arg reduction needed, so leave for second pa
121                         biguns = 1;
122                         goto MEDIUM;
123                 }
124                 if ( hx0 < 0x3e400000 ) {
125                         /* Too small.  cos x ~ 1. */
126                         v = *x;
127                         *y = 1.0;
128                         x += stridex;
129                         y += stridey;
130                         i = 0;
131                         if ( --n <= 0 )
132                                 break;
133                         goto LOOP0;
134                 }
135                 x0 = *x;
136                 py0 = y;
137                 x += stridex;
138                 y += stridey;
139                 i = 1;
140                 if ( --n <= 0 )
141                         break;
142
143 LOOP1: /* Get second arg, same as above. */
144                 xsb1 = HI(x);
145                 hx1 = xsb1 & ~0x80000000;
146                 if ( hx1 > 0x3fe921fb )
147                 {
148                         biguns = 2;
149                         goto MEDIUM;
150                 }
151                 if ( hx1 < 0x3e400000 )
152                 {
153                         v = *x;
154                         *y = 1.0;
155                         x += stridex;
156                         y += stridey;
157                         i = 1;
158                         if ( --n <= 0 )
159                                 break;
160                         goto LOOP1;
161                 }
162                 x1 = *x;
163                 py1 = y;
164                 x += stridex;
165                 y += stridey;
166                 i = 2;
167                 if ( --n <= 0 )
168                         break;
169
170 LOOP2: /* Get third arg, same as above. */
```



```
171                 xsb2 = HI(x);
172                 hx2 = xsb2 & ~0x80000000;
173                 if ( hx2 > 0x3fe921fb )
174                 {
175                         biguns = 3;
176                         goto MEDIUM;
177                 }
178                 if ( hx2 < 0x3e400000 )
179                 {
180                         v = *x;
181                         *y = 1.0;
182                         x += stridex;
183                         y += stridey;
184                         i = 2;
185                         if ( --n <= 0 )
186                                 break;
187                         goto LOOP2;
188                 }
189                 x2 = *x;
190                 py2 = y;
191
192                 /*
193                  * 0x3fc40000 = 5/32 ~ 0.15625
194                  * Get msb after subtraction.  Will be 1 only if
195                  * hx0 - 5/32 is negative.
196                  */
197                 i = ( hx0 - 0x3fc40000 ) >> 31;
198                 i |= ( ( hx1 - 0x3fc40000 ) >> 30 ) & 2;
199                 i |= ( ( hx2 - 0x3fc40000 ) >> 29 ) & 4;
200                 switch ( i )
201                 {
202                         double          a0, a1, a2, w0, w1, w2;
203                         double          t0, t1, t2, z0, z1, z2;
204                         unsigned        j0, j1, j2;
205
206                 case 0: /* All are > 5/32 */
207                         j0 = ( xsb0 + 0x4000 ) & 0xffff8000;
208                         j1 = ( xsb1 + 0x4000 ) & 0xffff8000;
209                         j2 = ( xsb2 + 0x4000 ) & 0xffff8000;
210                         HI(&t0) = j0;
211                         HI(&t1) = j1;
212                         HI(&t2) = j2;
213                         LO(&t0) = 0;
214                         LO(&t1) = 0;
215                         LO(&t2) = 0;
216                         x0 -= t0;
217                         x1 -= t1;
218                         x2 -= t2;
219                         z0 = x0 * x0;
220                         z1 = x1 * x1;
221                         z2 = x2 * x2;
222                         t0 = z0 * ( qq1 + z0 * qq2 );
223                         t1 = z1 * ( qq1 + z1 * qq2 );
224                         t2 = z2 * ( qq1 + z2 * qq2 );
225                         w0 = x0 * ( one + z0 * ( pp1 + z0 * pp2 ) );
226                         w1 = x1 * ( one + z1 * ( pp1 + z1 * pp2 ) );
227                         w2 = x2 * ( one + z2 * ( pp1 + z2 * pp2 ) );
228                         j0 = ( ( ( j0 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
229                         j1 = ( ( ( j1 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
230                         j2 = ( ( ( j2 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
231                         xsb0 = ( xsb0 >> 30 ) & 2;
232                         xsb1 = ( xsb1 >> 30 ) & 2;
233                         xsb2 = ( xsb2 >> 30 ) & 2;
234                         a0 = __vlibm_TBL_sincos_hi[j0+1]; /* cos_hi(t) */
235                         a1 = __vlibm_TBL_sincos_hi[j1+1];
236                         a2 = __vlibm_TBL_sincos_hi[j2+1];
```

```
                          /*   cos_lo(t)                          sin_hi(t) */
                        t0 = __vlibm_TBL_sincos_lo[j0+1] - ( __vlibm_TBL_sincos_
                        t1 = __vlibm_TBL_sincos_lo[j1+1] - ( __vlibm_TBL_sincos_
                        t2 = __vlibm_TBL_sincos_lo[j2+1] - ( __vlibm_TBL_sincos_

                        *py0 = a0 + t0;
                        *py1 = a1 + t1;
                        *py2 = a2 + t2;
                        break;

                case 1:
                        j1 = ( xsb1 + 0x4000 ) & 0xffff8000;
                        j2 = ( xsb2 + 0x4000 ) & 0xffff8000;
                        HI(&t1) = j1;
                        HI(&t2) = j2;
                        LO(&t1) = 0;
                        LO(&t2) = 0;
                        x1 -= t1;
                        x2 -= t2;
                        z0 = x0 * x0;
                        z1 = x1 * x1;
                        z2 = x2 * x2;
                        t0 = z0 * ( poly3[1] + z0 * poly4[1] );
                        t1 = z1 * ( qq1 + z1 * qq2 );
                        t2 = z2 * ( qq1 + z2 * qq2 );
                        t0 = z0 * ( poly1[1] + z0 * ( poly2[1] + t0 ) );
                        w1 = x1 * ( one + z1 * ( pp1 + z1 * pp2 ) );
                        w2 = x2 * ( one + z2 * ( pp1 + z2 * pp2 ) );
                        j1 = ( ( ( j1 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
                        j2 = ( ( ( j2 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
                        xsb1 = ( xsb1 >> 30 ) & 2;
                        xsb2 = ( xsb2 >> 30 ) & 2;
                        a1 = __vlibm_TBL_sincos_hi[j1+1];
                        a2 = __vlibm_TBL_sincos_hi[j2+1];
                        t1 = __vlibm_TBL_sincos_lo[j1+1] - ( __vlibm_TBL_sincos_
                        t2 = __vlibm_TBL_sincos_lo[j2+1] - ( __vlibm_TBL_sincos_
                        *py0 = one + t0;
                        *py1 = a1 + t1;
                        *py2 = a2 + t2;
                        break;

                case 2:
                        j0 = ( xsb0 + 0x4000 ) & 0xffff8000;
                        j2 = ( xsb2 + 0x4000 ) & 0xffff8000;
                        HI(&t0) = j0;
                        HI(&t2) = j2;
                        LO(&t0) = 0;
                        LO(&t2) = 0;
                        x0 -= t0;
                        x2 -= t2;
                        z0 = x0 * x0;
                        z1 = x1 * x1;
                        z2 = x2 * x2;
                        t0 = z0 * ( qq1 + z0 * qq2 );
                        t1 = z1 * ( poly3[1] + z1 * poly4[1] );
                        t2 = z2 * ( qq1 + z2 * qq2 );
                        w0 = x0 * ( one + z0 * ( pp1 + z0 * pp2 ) );
                        t1 = z1 * ( poly1[1] + z1 * ( poly2[1] + t1 ) );
                        w2 = x2 * ( one + z2 * ( pp1 + z2 * pp2 ) );
                        j0 = ( ( ( j0 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
                        j2 = ( ( ( j2 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
                        xsb0 = ( xsb0 >> 30 ) & 2;
                        xsb2 = ( xsb2 >> 30 ) & 2;
                        a0 = __vlibm_TBL_sincos_hi[j0+1];
                        a2 = __vlibm_TBL_sincos_hi[j2+1];
                        t0 = __vlibm_TBL_sincos_lo[j0+1] - ( __vlibm_TBL_sincos_
```



```
                        t2 = __vlibm_TBL_sincos_lo[j2+1] - ( __vlibm_TBL_sincos_
                        *py0 = a0 + t0;
                        *py1 = one + t1;
                        *py2 = a2 + t2;
                        break;

                case 3:
                        j2 = ( xsb2 + 0x4000 ) & 0xffff8000;
                        HI(&t2) = j2;
                        LO(&t2) = 0;
                        x2 -= t2;
                        z0 = x0 * x0;
                        z1 = x1 * x1;
                        z2 = x2 * x2;
                        t0 = z0 * ( poly3[1] + z0 * poly4[1] );
                        t1 = z1 * ( poly3[1] + z1 * poly4[1] );
                        t2 = z2 * ( qq1 + z2 * qq2 );
                        t0 = z0 * ( poly1[1] + z0 * ( poly2[1] + t0 ) );
                        t1 = z1 * ( poly1[1] + z1 * ( poly2[1] + t1 ) );
                        w2 = x2 * ( one + z2 * ( pp1 + z2 * pp2 ) );
                        j2 = ( ( ( j2 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
                        xsb2 = ( xsb2 >> 30 ) & 2;
                        a2 = __vlibm_TBL_sincos_hi[j2+1];
                        t2 = __vlibm_TBL_sincos_lo[j2+1] - ( __vlibm_TBL_sincos_
                        *py0 = one + t0;
                        *py1 = one + t1;
                        *py2 = a2 + t2;
                        break;

                case 4:
                        j0 = ( xsb0 + 0x4000 ) & 0xffff8000;
                        j1 = ( xsb1 + 0x4000 ) & 0xffff8000;
                        HI(&t0) = j0;
                        HI(&t1) = j1;
                        LO(&t0) = 0;
                        LO(&t1) = 0;
                        x0 -= t0;
                        x1 -= t1;
                        z0 = x0 * x0;
                        z1 = x1 * x1;
                        z2 = x2 * x2;
                        t0 = z0 * ( qq1 + z0 * qq2 );
                        t1 = z1 * ( qq1 + z1 * qq2 );
                        t2 = z2 * ( poly3[1] + z2 * poly4[1] );
                        w0 = x0 * ( one + z0 * ( pp1 + z0 * pp2 ) );
                        w1 = x1 * ( one + z1 * ( pp1 + z1 * pp2 ) );
                        t2 = z2 * ( poly1[1] + z2 * ( poly2[1] + t2 ) );
                        j0 = ( ( ( j0 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
                        j1 = ( ( ( j1 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
                        xsb0 = ( xsb0 >> 30 ) & 2;
                        xsb1 = ( xsb1 >> 30 ) & 2;
                        a0 = __vlibm_TBL_sincos_hi[j0+1];
                        a1 = __vlibm_TBL_sincos_hi[j1+1];
                        t0 = __vlibm_TBL_sincos_lo[j0+1] - ( __vlibm_TBL_sincos_
                        t1 = __vlibm_TBL_sincos_lo[j1+1] - ( __vlibm_TBL_sincos_
                        *py0 = a0 + t0;
                        *py1 = a1 + t1;
                        *py2 = one + t2;
                        break;

                case 5:
                        j1 = ( xsb1 + 0x4000 ) & 0xffff8000;
                        HI(&t1) = j1;
                        LO(&t1) = 0;
                        x1 -= t1;
                        z0 = x0 * x0;
```

```
369                         z1 = x1 * x1;
370                         z2 = x2 * x2;
371                         t0 = z0 * ( poly3[1] + z0 * poly4[1] );
372                         t1 = z1 * ( qq1 + z1 * qq2 );
373                         t2 = z2 * ( poly3[1] + z2 * poly4[1] );
374                         t0 = z0 * ( poly1[1] + z0 * ( poly2[1] + t0 ) );
375                         w1 = x1 * ( one + z1 * ( pp1 + z1 * pp2 ) );
376                         t2 = z2 * ( poly1[1] + z2 * ( poly2[1] + t2 ) );
377                         j1 = ( ( ( j1 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
378                         xsb1 = ( xsb1 >> 30 ) & 2;
379                         a1 = __vlibm_TBL_sincos_hi[j1+1];
380                         t1 = __vlibm_TBL_sincos_lo[j1+1] - ( __vlibm_TBL_sincos_
381                         *py0 = one + t0;
382                         *py1 = a1 + t1;
383                         *py2 = one + t2;
384                         break;

386                 case 6:
387                         j0 = ( xsb0 + 0x4000 ) & 0xffff8000;
388                         HI(&t0) = j0;
389                         LO(&t0) = 0;
390                         x0 -= t0;
391                         z0 = x0 * x0;
392                         z1 = x1 * x1;
393                         z2 = x2 * x2;
394                         t0 = z0 * ( qq1 + z0 * qq2 );
395                         t1 = z1 * ( poly3[1] + z1 * poly4[1] );
396                         t2 = z2 * ( poly3[1] + z2 * poly4[1] );
397                         w0 = x0 * ( one + z0 * ( pp1 + z0 * pp2 ) );
398                         t1 = z1 * ( poly1[1] + z1 * ( poly2[1] + t1 ) );
399                         t2 = z2 * ( poly1[1] + z2 * ( poly2[1] + t2 ) );
400                         j0 = ( ( ( j0 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
401                         xsb0 = ( xsb0 >> 30 ) & 2;
402                         a0 = __vlibm_TBL_sincos_hi[j0+1];
403                         t0 = __vlibm_TBL_sincos_lo[j0+1] - ( __vlibm_TBL_sincos_
404                         *py0 = a0 + t0;
405                         *py1 = one + t1;
406                         *py2 = one + t2;
407                         break;

409                 case 7: /* All are < 5/32 */
410                         z0 = x0 * x0;
411                         z1 = x1 * x1;
412                         z2 = x2 * x2;
413                         t0 = z0 * ( poly3[1] + z0 * poly4[1] );
414                         t1 = z1 * ( poly3[1] + z1 * poly4[1] );
415                         t2 = z2 * ( poly3[1] + z2 * poly4[1] );
416                         t0 = z0 * ( poly1[1] + z0 * ( poly2[1] + t0 ) );
417                         t1 = z1 * ( poly1[1] + z1 * ( poly2[1] + t1 ) );
418                         t2 = z2 * ( poly1[1] + z2 * ( poly2[1] + t2 ) );
419                         *py0 = one + t0;
420                         *py1 = one + t1;
421                         *py2 = one + t2;
422                         break;
423                 }

425                 x += stridex;
426                 y += stridey;
427                 i = 0;
428         } while ( --n > 0 ); /* END MAIN LOOP */

430         /*
431          * CLEAN UP last 0, 1, or 2 elts.
432          */
433         if ( i > 0 ) /* Clean up elts at tail.  i < 3. */
434         {
```



```
435                 double          a0, a1, w0, w1;
436                 double          t0, t1, z0, z1;
437                 unsigned        j0, j1;

439                 if ( i > 1 )
440                 {
441                         if ( hx1 < 0x3fc40000 )
442                         {
443                                 z1 = x1 * x1;
444                                 t1 = z1 * ( poly3[1] + z1 * poly4[1] );
445                                 t1 = z1 * ( poly1[1] + z1 * ( poly2[1] + t1 ) );
446                                 t1 = one + t1;
447                                 *py1 = t1;
448                         }
449                         else
450                         {
451                                 j1 = ( xsb1 + 0x4000 ) & 0xffff8000;
452                                 HI(&t1) = j1;
453                                 LO(&t1) = 0;
454                                 x1 -= t1;
455                                 z1 = x1 * x1;
456                                 t1 = z1 * ( qq1 + z1 * qq2 );
457                                 w1 = x1 * ( one + z1 * ( pp1 + z1 * pp2 ) );
458                                 j1 = ( ( ( j1 & ~0x80000000 ) - 0x3fc40000 ) >>
459                                 xsb1 = ( xsb1 >> 30 ) & 2;
460                                 a1 = __vlibm_TBL_sincos_hi[j1+1];
461                                 t1 = __vlibm_TBL_sincos_lo[j1+1]
462                                         - ( __vlibm_TBL_sincos_hi[j1+xsb1]*w1 -
463                                 *py1 = a1 + t1;
464                         }
465                 }
466                 if ( hx0 < 0x3fc40000 )
467                 {
468                         z0 = x0 * x0;
469                         t0 = z0 * ( poly3[1] + z0 * poly4[1] );
470                         t0 = z0 * ( poly1[1] + z0 * ( poly2[1] + t0 ) );
471                         t0 = one + t0;
472                         *py0 = t0;
473                 }
474                 else
475                 {
476                         j0 = ( xsb0 + 0x4000 ) & 0xffff8000;
477                         HI(&t0) = j0;
478                         LO(&t0) = 0;
479                         x0 -= t0;
480                         z0 = x0 * x0;
481                         t0 = z0 * ( qq1 + z0 * qq2 );
482                         w0 = x0 * ( one + z0 * ( pp1 + z0 * pp2 ) );
483                         j0 = ( ( ( j0 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
484                         xsb0 = ( xsb0 >> 30 ) & 2;
485                         a0 = __vlibm_TBL_sincos_hi[j0+1];
486                         t0 = __vlibm_TBL_sincos_lo[j0+1] - ( __vlibm_TBL_sincos_
487                         *py0 = a0 + t0;
488                 }
489         } /* END CLEAN UP */

491         return;

493         /*
494          * Take care of BIGUNS.
495          *
496          * We have jumped here in the middle of processing after having
497          * encountered a medium range argument.  Therefore things are in a
498          * bit of a tizzy.
499          */
```

```
MEDIUM:

        x0_or_one[1] = 1.0;
        x1_or_one[1] = 1.0;
        x2_or_one[1] = 1.0;
        x0_or_one[3] = -1.0;
        x1_or_one[3] = -1.0;
        x2_or_one[3] = -1.0;
        y0_or_zero[1] = 0.0;
        y1_or_zero[1] = 0.0;
        y2_or_zero[1] = 0.0;
        y0_or_zero[3] = 0.0;
        y1_or_zero[3] = 0.0;
        y2_or_zero[3] = 0.0;

        if ( biguns == 3 )
        {
                biguns = 0;
                xsb0 = xsb0 >> 31;
                xsb1 = xsb1 >> 31;
                goto loop2;
        }
        else if ( biguns == 2 )
        {
                xsb0 = xsb0 >> 31;
                biguns = 0;
                goto loop1;
        }
        biguns = 0;

        do
        {
                double          fn0, fn1, fn2, a0, a1, a2, w0, w1, w2, y0, y1, y
                unsigned        hx;
                int                     n0, n1, n2;

                /*
                 * Find 3 more to work on: Not already done, not too big.
                 */

loop0:
                hx = HI(x);
                xsb0 = hx >> 31;
                hx &= ~0x80000000;
                if ( hx > 0x413921fb ) /* (1.6471e+06) Too big: leave it. */
                {
                        if ( hx >= 0x7ff00000 ) /* Inf or NaN */
                        {
                                x0 = *x;
                                *y = x0 - x0;
                        }
                        else
                                biguns = 1;
                        x += stridex;
                        y += stridey;
                        i = 0;
                        if ( --n <= 0 )
                                break;
                        goto loop0;
                }
                x0 = *x;
                py0 = y;
                x += stridex;
                y += stridey;
                i = 1;
                if ( --n <= 0 )
```



```
                        break;

loop1:
                hx = HI(x);
                xsb1 = hx >> 31;
                hx &= ~0x80000000;
                if ( hx > 0x413921fb )
                {
                        if ( hx >= 0x7ff00000 )
                        {
                                x1 = *x;
                                *y = x1 - x1;
                        }
                        else
                                biguns = 1;
                        x += stridex;
                        y += stridey;
                        i = 1;
                        if ( --n <= 0 )
                                break;
                        goto loop1;
                }
                x1 = *x;
                py1 = y;
                x += stridex;
                y += stridey;
                i = 2;
                if ( --n <= 0 )
                        break;

loop2:
                hx = HI(x);
                xsb2 = hx >> 31;
                hx &= ~0x80000000;
                if ( hx > 0x413921fb )
                {
                        if ( hx >= 0x7ff00000 )
                        {
                                x2 = *x;
                                *y = x2 - x2;
                        }
                        else
                                biguns = 1;
                        x += stridex;
                        y += stridey;
                        i = 2;
                        if ( --n <= 0 )
                                break;
                        goto loop2;
                }
                x2 = *x;
                py2 = y;

                n0 = (int) ( x0 * invpio2 + half[xsb0] );
                n1 = (int) ( x1 * invpio2 + half[xsb1] );
                n2 = (int) ( x2 * invpio2 + half[xsb2] );
                fn0 = (double) n0;
                fn1 = (double) n1;
                fn2 = (double) n2;
                n0 = (n0 + 1) & 3; /* Add 1 (before the mod) to make sin into co
                n1 = (n1 + 1) & 3;
                n2 = (n2 + 1) & 3;
                a0 = x0 - fn0 * pio2_1;
                a1 = x1 - fn1 * pio2_1;
                a2 = x2 - fn2 * pio2_1;
                w0 = fn0 * pio2_2;
```

```
633                 w1 = fn1 * pio2_2;
634                 w2 = fn2 * pio2_2;
635                 x0 = a0 - w0;
636                 x1 = a1 - w1;
637                 x2 = a2 - w2;
638                 y0 = ( a0 - x0 ) - w0;
639                 y1 = ( a1 - x1 ) - w1;
640                 y2 = ( a2 - x2 ) - w2;
641                 a0 = x0;
642                 a1 = x1;
643                 a2 = x2;
644                 w0 = fn0 * pio2_3 - y0;
645                 w1 = fn1 * pio2_3 - y1;
646                 w2 = fn2 * pio2_3 - y2;
647                 x0 = a0 - w0;
648                 x1 = a1 - w1;
649                 x2 = a2 - w2;
650                 y0 = ( a0 - x0 ) - w0;
651                 y1 = ( a1 - x1 ) - w1;
652                 y2 = ( a2 - x2 ) - w2;
653                 a0 = x0;
654                 a1 = x1;
655                 a2 = x2;
656                 w0 = fn0 * pio2_3t - y0;
657                 w1 = fn1 * pio2_3t - y1;
658                 w2 = fn2 * pio2_3t - y2;
659                 x0 = a0 - w0;
660                 x1 = a1 - w1;
661                 x2 = a2 - w2;
662                 y0 = ( a0 - x0 ) - w0;
663                 y1 = ( a1 - x1 ) - w1;
664                 y2 = ( a2 - x2 ) - w2;
665                 xsb0 = HI(&x0);
666                 i = ( ( xsb0 & ~0x80000000 ) - thresh[n0&1] ) >> 31;
667                 xsb1 = HI(&x1);
668                 i |= ( ( ( xsb1 & ~0x80000000 ) - thresh[n1&1] ) >> 30 ) & 2;
669                 xsb2 = HI(&x2);
670                 i |= ( ( ( xsb2 & ~0x80000000 ) - thresh[n2&1] ) >> 29 ) & 4;
671                 switch ( i )
672                 {
673                         double          t0, t1, t2, z0, z1, z2;
674                         unsigned        j0, j1, j2;

676                 case 0:
677                         j0 = ( xsb0 + 0x4000 ) & 0xffff8000;
678                         j1 = ( xsb1 + 0x4000 ) & 0xffff8000;
679                         j2 = ( xsb2 + 0x4000 ) & 0xffff8000;
680                         HI(&t0) = j0;
681                         HI(&t1) = j1;
682                         HI(&t2) = j2;
683                         LO(&t0) = 0;
684                         LO(&t1) = 0;
685                         LO(&t2) = 0;
686                         x0 = ( x0 - t0 ) + y0;
687                         x1 = ( x1 - t1 ) + y1;
688                         x2 = ( x2 - t2 ) + y2;
689                         z0 = x0 * x0;
690                         z1 = x1 * x1;
691                         z2 = x2 * x2;
692                         t0 = z0 * ( qq1 + z0 * qq2 );
693                         t1 = z1 * ( qq1 + z1 * qq2 );
694                         t2 = z2 * ( qq1 + z2 * qq2 );
695                         w0 = x0 * ( one + z0 * ( pp1 + z0 * pp2 ) );
696                         w1 = x1 * ( one + z1 * ( pp1 + z1 * pp2 ) );
697                         w2 = x2 * ( one + z2 * ( pp1 + z2 * pp2 ) );
698                         j0 = ( ( ( j0 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
```



```
699                         j1 = ( ( ( j1 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
700                         j2 = ( ( ( j2 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
701                         xsb0 = ( xsb0 >> 30 ) & 2;
702                         xsb1 = ( xsb1 >> 30 ) & 2;
703                         xsb2 = ( xsb2 >> 30 ) & 2;
704                         n0 ^= ( xsb0 & ~( n0 << 1 ) );
705                         n1 ^= ( xsb1 & ~( n1 << 1 ) );
706                         n2 ^= ( xsb2 & ~( n2 << 1 ) );
707                         xsb0 |= 1;
708                         xsb1 |= 1;
709                         xsb2 |= 1;
710                         a0 = __vlibm_TBL_sincos_hi[j0+n0];
711                         a1 = __vlibm_TBL_sincos_hi[j1+n1];
712                         a2 = __vlibm_TBL_sincos_hi[j2+n2];
713                         t0 = ( __vlibm_TBL_sincos_hi[j0+((n0+xsb0)&3)] * w0 + a0
714                         t1 = ( __vlibm_TBL_sincos_hi[j1+((n1+xsb1)&3)] * w1 + a1
715                         t2 = ( __vlibm_TBL_sincos_hi[j2+((n2+xsb2)&3)] * w2 + a2
716                         *py0 = ( a0 + t0 );
717                         *py1 = ( a1 + t1 );
718                         *py2 = ( a2 + t2 );
719                         break;

721                 case 1:
722                         j0 = n0 & 1;
723                         j1 = ( xsb1 + 0x4000 ) & 0xffff8000;
724                         j2 = ( xsb2 + 0x4000 ) & 0xffff8000;
725                         HI(&t1) = j1;
726                         HI(&t2) = j2;
727                         LO(&t1) = 0;
728                         LO(&t2) = 0;
729                         x0_or_one[0] = x0;
730                         x0_or_one[2] = -x0;
731                         y0_or_zero[0] = y0;
732                         y0_or_zero[2] = -y0;
733                         x1 = ( x1 - t1 ) + y1;
734                         x2 = ( x2 - t2 ) + y2;
735                         z0 = x0 * x0;
736                         z1 = x1 * x1;
737                         z2 = x2 * x2;
738                         t0 = z0 * ( poly3[j0] + z0 * poly4[j0] );
739                         t1 = z1 * ( qq1 + z1 * qq2 );
740                         t2 = z2 * ( qq1 + z2 * qq2 );
741                         t0 = z0 * ( poly1[j0] + z0 * ( poly2[j0] + t0 ) );
742                         w1 = x1 * ( one + z1 * ( pp1 + z1 * pp2 ) );
743                         w2 = x2 * ( one + z2 * ( pp1 + z2 * pp2 ) );
744                         j1 = ( ( ( j1 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
745                         j2 = ( ( ( j2 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
746                         xsb1 = ( xsb1 >> 30 ) & 2;
747                         xsb2 = ( xsb2 >> 30 ) & 2;
748                         n1 ^= ( xsb1 & ~( n1 << 1 ) );
749                         n2 ^= ( xsb2 & ~( n2 << 1 ) );
750                         xsb1 |= 1;
751                         xsb2 |= 1;
752                         a1 = __vlibm_TBL_sincos_hi[j1+n1];
753                         a2 = __vlibm_TBL_sincos_hi[j2+n2];
754                         t0 = x0_or_one[n0] + ( y0_or_zero[n0] + x0_or_one[n0] *
755                         t1 = ( __vlibm_TBL_sincos_hi[j1+((n1+xsb1)&3)] * w1 + a1
756                         t2 = ( __vlibm_TBL_sincos_hi[j2+((n2+xsb2)&3)] * w2 + a2
757                         *py0 = t0;
758                         *py1 = ( a1 + t1 );
759                         *py2 = ( a2 + t2 );
760                         break;

762                 case 2:
763                         j0 = ( xsb0 + 0x4000 ) & 0xffff8000;
764                         j1 = n1 & 1;
```

```
765                     j2 = ( xsb2 + 0x4000 ) & 0xffff8000;
766                     HI(&t0) = j0;
767                     HI(&t2) = j2;
768                     LO(&t0) = 0;
769                     LO(&t2) = 0;
770                     x1_or_one[0] = x1;
771                     x1_or_one[2] = -x1;
772                     x0 = ( x0 - t0 ) + y0;
773                     y1_or_zero[0] = y1;
774                     y1_or_zero[2] = -y1;
775                     x2 = ( x2 - t2 ) + y2;
776                     z0 = x0 * x0;
777                     z1 = x1 * x1;
778                     z2 = x2 * x2;
779                     t0 = z0 * ( qq1 + z0 * qq2 );
780                     t1 = z1 * ( poly3[j1] + z1 * poly4[j1] );
781                     t2 = z2 * ( qq1 + z2 * qq2 );
782                     w0 = x0 * ( one + z0 * ( pp1 + z0 * pp2 ) );
783                     t1 = z1 * ( poly1[j1] + z1 * ( poly2[j1] + t1 ) );
784                     w2 = x2 * ( one + z2 * ( pp1 + z2 * pp2 ) );
785                     j0 = ( ( ( j0 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
786                     j2 = ( ( ( j2 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
787                     xsb0 = ( xsb0 >> 30 ) & 2;
788                     xsb2 = ( xsb2 >> 30 ) & 2;
789                     n0 ^= ( xsb0 & ~( n0 << 1 ) );
790                     n2 ^= ( xsb2 & ~( n2 << 1 ) );
791                     xsb0 |= 1;
792                     xsb2 |= 1;
793                     a0 = __vlibm_TBL_sincos_hi[j0+n0];
794                     a2 = __vlibm_TBL_sincos_hi[j2+n2];
795                     t0 = ( __vlibm_TBL_sincos_hi[j0+((n0+xsb0)&3)] * w0 + a0
796                     t1 = x1_or_one[n1] + ( y1_or_zero[n1] + x1_or_one[n1] *
797                     t2 = ( __vlibm_TBL_sincos_hi[j2+((n2+xsb2)&3)] * w2 + a2
798                     *py0 = ( a0 + t0 );
799                     *py1 = t1;
800                     *py2 = ( a2 + t2 );
801                     break;

803             case 3:
804                     j0 = n0 & 1;
805                     j1 = n1 & 1;
806                     j2 = ( xsb2 + 0x4000 ) & 0xffff8000;
807                     HI(&t2) = j2;
808                     LO(&t2) = 0;
809                     x0_or_one[0] = x0;
810                     x0_or_one[2] = -x0;
811                     x1_or_one[0] = x1;
812                     x1_or_one[2] = -x1;
813                     y0_or_zero[0] = y0;
814                     y0_or_zero[2] = -y0;
815                     y1_or_zero[0] = y1;
816                     y1_or_zero[2] = -y1;
817                     x2 = ( x2 - t2 ) + y2;
818                     z0 = x0 * x0;
819                     z1 = x1 * x1;
820                     z2 = x2 * x2;
821                     t0 = z0 * ( poly3[j0] + z0 * poly4[j0] );
822                     t1 = z1 * ( poly3[j1] + z1 * poly4[j1] );
823                     t2 = z2 * ( qq1 + z2 * qq2 );
824                     t0 = z0 * ( poly1[j0] + z0 * ( poly2[j0] + t0 ) );
825                     t1 = z1 * ( poly1[j1] + z1 * ( poly2[j1] + t1 ) );
826                     w2 = x2 * ( one + z2 * ( pp1 + z2 * pp2 ) );
827                     j2 = ( ( ( j2 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
828                     xsb2 = ( xsb2 >> 30 ) & 2;
829                     n2 ^= ( xsb2 & ~( n2 << 1 ) );
830                     xsb2 |= 1;
```



```
831                     a2 = __vlibm_TBL_sincos_hi[j2+n2];
832                     t0 = x0_or_one[n0] + ( y0_or_zero[n0] + x0_or_one[n0] *
833                     t1 = x1_or_one[n1] + ( y1_or_zero[n1] + x1_or_one[n1] *
834                     t2 = ( __vlibm_TBL_sincos_hi[j2+((n2+xsb2)&3)] * w2 + a2
835                     *py0 = t0;
836                     *py1 = t1;
837                     *py2 = ( a2 + t2 );
838                     break;

840             case 4:
841                     j0 = ( xsb0 + 0x4000 ) & 0xffff8000;
842                     j1 = ( xsb1 + 0x4000 ) & 0xffff8000;
843                     j2 = n2 & 1;
844                     HI(&t0) = j0;
845                     HI(&t1) = j1;
846                     LO(&t0) = 0;
847                     LO(&t1) = 0;
848                     x2_or_one[0] = x2;
849                     x2_or_one[2] = -x2;
850                     x0 = ( x0 - t0 ) + y0;
851                     x1 = ( x1 - t1 ) + y1;
852                     y2_or_zero[0] = y2;
853                     y2_or_zero[2] = -y2;
854                     z0 = x0 * x0;
855                     z1 = x1 * x1;
856                     z2 = x2 * x2;
857                     t0 = z0 * ( qq1 + z0 * qq2 );
858                     t1 = z1 * ( qq1 + z1 * qq2 );
859                     t2 = z2 * ( poly3[j2] + z2 * poly4[j2] );
860                     w0 = x0 * ( one + z0 * ( pp1 + z0 * pp2 ) );
861                     w1 = x1 * ( one + z1 * ( pp1 + z1 * pp2 ) );
862                     t2 = z2 * ( poly1[j2] + z2 * ( poly2[j2] + t2 ) );
863                     j0 = ( ( ( j0 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
864                     j1 = ( ( ( j1 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
865                     xsb0 = ( xsb0 >> 30 ) & 2;
866                     xsb1 = ( xsb1 >> 30 ) & 2;
867                     n0 ^= ( xsb0 & ~( n0 << 1 ) );
868                     n1 ^= ( xsb1 & ~( n1 << 1 ) );
869                     xsb0 |= 1;
870                     xsb1 |= 1;
871                     a0 = __vlibm_TBL_sincos_hi[j0+n0];
872                     a1 = __vlibm_TBL_sincos_hi[j1+n1];
873                     t0 = ( __vlibm_TBL_sincos_hi[j0+((n0+xsb0)&3)] * w0 + a0
874                     t1 = ( __vlibm_TBL_sincos_hi[j1+((n1+xsb1)&3)] * w1 + a1
875                     t2 = x2_or_one[n2] + ( y2_or_zero[n2] + x2_or_one[n2] *
876                     *py0 = ( a0 + t0 );
877                     *py1 = ( a1 + t1 );
878                     *py2 = t2;
879                     break;

881             case 5:
882                     j0 = n0 & 1;
883                     j1 = ( xsb1 + 0x4000 ) & 0xffff8000;
884                     j2 = n2 & 1;
885                     HI(&t1) = j1;
886                     LO(&t1) = 0;
887                     x0_or_one[0] = x0;
888                     x0_or_one[2] = -x0;
889                     x2_or_one[0] = x2;
890                     x2_or_one[2] = -x2;
891                     y0_or_zero[0] = y0;
892                     y0_or_zero[2] = -y0;
893                     x1 = ( x1 - t1 ) + y1;
894                     y2_or_zero[0] = y2;
895                     y2_or_zero[2] = -y2;
896                     z0 = x0 * x0;
```

```
897                         z1 = x1 * x1;
898                         z2 = x2 * x2;
899                         t0 = z0 * ( poly3[j0] + z0 * poly4[j0] );
900                         t1 = z1 * ( qq1 + z1 * qq2 );
901                         t2 = z2 * ( poly3[j2] + z2 * poly4[j2] );
902                         t0 = z0 * ( poly1[j0] + z0 * ( poly2[j0] + t0 ) );
903                         w1 = x1 * ( one + z1 * ( pp1 + z1 * pp2 ) );
904                         t2 = z2 * ( poly1[j2] + z2 * ( poly2[j2] + t2 ) );
905                         j1 = ( ( ( j1 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
906                         xsb1 = ( xsb1 >> 30 ) & 2;
907                         n1 ^= ( xsb1 & ~( n1 << 1 ) );
908                         xsb1 |= 1;
909                         a1 = __vlibm_TBL_sincos_hi[j1+n1];
910                         t0 = x0_or_one[n0] + ( y0_or_zero[n0] + x0_or_one[n0] *
911                         t1 = ( __vlibm_TBL_sincos_hi[j1+((n1+xsb1)&3)] * w1 + a1
912                         t2 = x2_or_one[n2] + ( y2_or_zero[n2] + x2_or_one[n2] *
913                         *py0 = t0;
914                         *py1 = ( a1 + t1 );
915                         *py2 = t2;
916                         break;

918                 case 6:
919                         j0 = ( xsb0 + 0x4000 ) & 0xffff8000;
920                         j1 = n1 & 1;
921                         j2 = n2 & 1;
922                         HI(&t0) = j0;
923                         LO(&t0) = 0;
924                         x1_or_one[0] = x1;
925                         x1_or_one[2] = -x1;
926                         x2_or_one[0] = x2;
927                         x2_or_one[2] = -x2;
928                         x0 = ( x0 - t0 ) + y0;
929                         y1_or_zero[0] = y1;
930                         y1_or_zero[2] = -y1;
931                         y2_or_zero[0] = y2;
932                         y2_or_zero[2] = -y2;
933                         z0 = x0 * x0;
934                         z1 = x1 * x1;
935                         z2 = x2 * x2;
936                         t0 = z0 * ( qq1 + z0 * qq2 );
937                         t1 = z1 * ( poly3[j1] + z1 * poly4[j1] );
938                         t2 = z2 * ( poly3[j2] + z2 * poly4[j2] );
939                         w0 = x0 * ( one + z0 * ( pp1 + z0 * pp2 ) );
940                         t1 = z1 * ( poly1[j1] + z1 * ( poly2[j1] + t1 ) );
941                         t2 = z2 * ( poly1[j2] + z2 * ( poly2[j2] + t2 ) );
942                         j0 = ( ( ( j0 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
943                         xsb0 = ( xsb0 >> 30 ) & 2;
944                         n0 ^= ( xsb0 & ~( n0 << 1 ) );
945                         xsb0 |= 1;
946                         a0 = __vlibm_TBL_sincos_hi[j0+n0];
947                         t0 = ( __vlibm_TBL_sincos_hi[j0+((n0+xsb0)&3)] * w0 + a0
948                         t1 = x1_or_one[n1] + ( y1_or_zero[n1] + x1_or_one[n1] *
949                         t2 = x2_or_one[n2] + ( y2_or_zero[n2] + x2_or_one[n2] *
950                         *py0 = ( a0 + t0 );
951                         *py1 = t1;
952                         *py2 = t2;
953                         break;

955                 case 7:
956                         j0 = n0 & 1;
957                         j1 = n1 & 1;
958                         j2 = n2 & 1;
959                         x0_or_one[0] = x0;
960                         x0_or_one[2] = -x0;
961                         x1_or_one[0] = x1;
962                         x1_or_one[2] = -x1;
```



```
963                         x2_or_one[0] = x2;
964                         x2_or_one[2] = -x2;
965                         y0_or_zero[0] = y0;
966                         y0_or_zero[2] = -y0;
967                         y1_or_zero[0] = y1;
968                         y1_or_zero[2] = -y1;
969                         y2_or_zero[0] = y2;
970                         y2_or_zero[2] = -y2;
971                         z0 = x0 * x0;
972                         z1 = x1 * x1;
973                         z2 = x2 * x2;
974                         t0 = z0 * ( poly3[j0] + z0 * poly4[j0] );
975                         t1 = z1 * ( poly3[j1] + z1 * poly4[j1] );
976                         t2 = z2 * ( poly3[j2] + z2 * poly4[j2] );
977                         t0 = z0 * ( poly1[j0] + z0 * ( poly2[j0] + t0 ) );
978                         t1 = z1 * ( poly1[j1] + z1 * ( poly2[j1] + t1 ) );
979                         t2 = z2 * ( poly1[j2] + z2 * ( poly2[j2] + t2 ) );
980                         t0 = x0_or_one[n0] + ( y0_or_zero[n0] + x0_or_one[n0] *
981                         t1 = x1_or_one[n1] + ( y1_or_zero[n1] + x1_or_one[n1] *
982                         t2 = x2_or_one[n2] + ( y2_or_zero[n2] + x2_or_one[n2] *
983                         *py0 = t0;
984                         *py1 = t1;
985                         *py2 = t2;
986                         break;
987                 }

989                 x += stridex;
990                 y += stridey;
991                 i = 0;
992         } while ( --n > 0 );

994         if ( i > 0 )
995         {
996                 double          fn0, fn1, a0, a1, w0, w1, y0, y1;
997                 double          t0, t1, z0, z1;
998                 unsigned        j0, j1;
999                 int                     n0, n1;

1001                if ( i > 1 )
1002                {
1003                        n1 = (int) ( x1 * invpio2 + half[xsb1] );
1004                        fn1 = (double) n1;
1005                        n1 = (n1 + 1) & 3; /* Add 1 (before the mod) to make sin
1006                        a1 = x1 - fn1 * pio2_1;
1007                        w1 = fn1 * pio2_2;
1008                        x1 = a1 - w1;
1009                        y1 = ( a1 - x1 ) - w1;
1010                        a1 = x1;
1011                        w1 = fn1 * pio2_3 - y1;
1012                        x1 = a1 - w1;
1013                        y1 = ( a1 - x1 ) - w1;
1014                        a1 = x1;
1015                        w1 = fn1 * pio2_3t - y1;
1016                        x1 = a1 - w1;
1017                        y1 = ( a1 - x1 ) - w1;
1018                        xsb1 = HI(&x1);
1019                        if ( ( xsb1 & ~0x80000000 ) < thresh[n1&1] )
1020                        {
1021                                j1 = n1 & 1;
1022                                x1_or_one[0] = x1;
1023                                x1_or_one[2] = -x1;
1024                                y1_or_zero[0] = y1;
1025                                y1_or_zero[2] = -y1;
1026                                z1 = x1 * x1;
1027                                t1 = z1 * ( poly3[j1] + z1 * poly4[j1] );
1028                                t1 = z1 * ( poly1[j1] + z1 * ( poly2[j1] + t1 )
```

```
1029                                t1 = x1_or_one[n1] + ( y1_or_zero[n1] + x1_or_on
1030                                *py1 = t1;
1031                        }
1032                        else
1033                        {
1034                                j1 = ( xsb1 + 0x4000 ) & 0xffff8000;
1035                                HI(&t1) = j1;
1036                                LO(&t1) = 0;
1037                                x1 = ( x1 - t1 ) + y1;
1038                                z1 = x1 * x1;
1039                                t1 = z1 * ( qq1 + z1 * qq2 );
1040                                w1 = x1 * ( one + z1 * ( pp1 + z1 * pp2 ) );
1041                                j1 = ( ( ( j1 & ~0x80000000 ) - 0x3fc40000 ) >>
1042                                xsb1 = ( xsb1 >> 30 ) & 2;
1043                                n1 ^= ( xsb1 & ~( n1 << 1 ) );
1044                                xsb1 |= 1;
1045                                a1 = __vlibm_TBL_sincos_hi[j1+n1];
1046                                t1 = ( __vlibm_TBL_sincos_hi[j1+((n1+xsb1)&3)] *
1047                                *py1 = ( a1 + t1 );
1048                        }
1049                }
1050                n0 = (int) ( x0 * invpio2 + half[xsb0] );
1051                fn0 = (double) n0;
1052                n0 = (n0 + 1) & 3; /* Add 1 (before the mod) to make sin into co
1053                a0 = x0 - fn0 * pio2_1;
1054                w0 = fn0 * pio2_2;
1055                x0 = a0 - w0;
1056                y0 = ( a0 - x0 ) - w0;
1057                a0 = x0;
1058                w0 = fn0 * pio2_3 - y0;
1059                x0 = a0 - w0;
1060                y0 = ( a0 - x0 ) - w0;
1061                a0 = x0;
1062                w0 = fn0 * pio2_3t - y0;
1063                x0 = a0 - w0;
1064                y0 = ( a0 - x0 ) - w0;
1065                xsb0 = HI(&x0);
1066                if ( ( xsb0 & ~0x80000000 ) < thresh[n0&1] )
1067                {
1068                        j0 = n0 & 1;
1069                        x0_or_one[0] = x0;
1070                        x0_or_one[2] = -x0;
1071                        y0_or_zero[0] = y0;
1072                        y0_or_zero[2] = -y0;
1073                        z0 = x0 * x0;
1074                        t0 = z0 * ( poly3[j0] + z0 * poly4[j0] );
1075                        t0 = z0 * ( poly1[j0] + z0 * ( poly2[j0] + t0 ) );
1076                        t0 = x0_or_one[n0] + ( y0_or_zero[n0] + x0_or_one[n0] *
1077                        *py0 = t0;
1078                }
1079                else
1080                {
1081                        j0 = ( xsb0 + 0x4000 ) & 0xffff8000;
1082                        HI(&t0) = j0;
1083                        LO(&t0) = 0;
1084                        x0 = ( x0 - t0 ) + y0;
1085                        z0 = x0 * x0;
1086                        t0 = z0 * ( qq1 + z0 * qq2 );
1087                        w0 = x0 * ( one + z0 * ( pp1 + z0 * pp2 ) );
1088                        j0 = ( ( ( j0 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
1089                        xsb0 = ( xsb0 >> 30 ) & 2;
1090                        n0 ^= ( xsb0 & ~( n0 << 1 ) );
1091                        xsb0 |= 1;
1092                        a0 = __vlibm_TBL_sincos_hi[j0+n0];
1093                        t0 = ( __vlibm_TBL_sincos_hi[j0+((n0+xsb0)&3)] * w0 + a0
1094                        *py0 = ( a0 + t0 );
```



```
1095                }
1096        }

1098        if ( biguns )
1099                __vlibm_vcos_big( nsave, xsave, sxsave, ysave, sysave, 0x413921f
1100 }
```

```
**********************************************************
   28847 Thu Oct  9 19:48:55 2014
new/usr/src/lib/libmvec/common/__vsin.c
Revert "remove unused v from libmvec"
This reverts commit e853d278ee4b7f2a8c2117cc598cfc68b4e3f29b.
fix system-library-math.mf
update libm manifests
14071:dece9aafe99a - fix build problems on sparc
remove unused v from libmvec
fix for patch09 - use __GNU_UNUSED
patch12 - math.h: Align things with GCC
patch11 - added LIBM man pages
patch09 - update libmvec: fix build issues by gcc46
patch08 - libmvec: fixed compilation issues after updates
patch01 - 693 import Sun Devpro Math Library
Revert "remove unused v from libmvec"
This reverts commit e853d278ee4b7f2a8c2117cc598cfc68b4e3f29b.
fix system-library-math.mf
update libm manifests
14071:dece9aafe99a - fix build problems on sparc
remove unused v from libmvec
fix for patch09 - use __GNU_UNUSED
patch12 - math.h: Align things with GCC
patch11 - added LIBM man pages
patch09 - update libmvec: fix build issues by gcc46
patch08 - libmvec: fixed compilation issues after updates
patch01 - 693 import Sun Devpro Math Library
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" 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.
 */

#include <sys/isa_defs.h>
#include <sys/ccompile.h>

#ifdef _LITTLE_ENDIAN
#define HI(x)   *(1+(int*)x)
#define LO(x)   *(unsigned*)x
#else
#define HI(x)   *(int*)x
#define LO(x)   *(1+(unsigned*)x)
```



```
#endif

#ifdef __RESTRICT
#define restrict _Restrict
#else
#define restrict
#endif

extern const double __vlibm_TBL_sincos_hi[], __vlibm_TBL_sincos_lo[];

static const double
        half[2] = { 0.5, -0.5 },
        one             = 1.0,
        invpio2 = 0.6366197723675813430755e-00,
        pio2_1  = 1.57079632673412561416e-00,
        pio2_2  = 6.07710050630396597659e-11,
        pio2_3  = 2.02226248711166455796e-21,
        pio2_3t = 8.47842766036889964395e-32,
        pp1             = -1.66666666660576046527626394313498255467e-0001,
        pp2             =  8.33326120969096312671837656614618094444e-0003,
        qq1             = -4.99999999977710986407023955908711557870e-0001,
        qq2             =  4.16665486385721935064505588101884208958e-0002,
        poly1[2]= { -1.66666666666629669805215138920301589656e-0001,
                                -4.99999999999931701464060878888294524481e-0001
        poly2[2]= {  8.33333332390951295683993455280336376663e-0003,
                                 4.16666666394861917535640593963708222319e-0002
        poly3[2]= { -1.98412623799797669279155177823009840396e-0004,
                                -1.38888552656142867832756687736851681462e-0003
        poly4[2]= {  2.75340362485427723764998762284833035111e-0006,
                                 2.47851942368146079661812828945453052475e-0005

static const unsigned thresh[2] = { 0x3fc90000, 0x3fc40000 };

/* Don't __ the following; acomp will handle it */
extern double fabs( double );
extern void __vlibm_vsin_big( int, double *, int, double *, int, int );

void
__vsin( int n, double * restrict x, int stridex, double * restrict y,
        int stridey )
{
        double          x0_or_one[4], x1_or_one[4], x2_or_one[4];
        double          y0_or_zero[4], y1_or_zero[4], y2_or_zero[4];
        double          x0, x1, x2, *py0 = 0, *py1 = 0, *py2, *xsave, *ysave;
        unsigned        hx0, hx1, hx2, xsb0, xsb1 = 0, xsb2;
        int             i, biguns, nsave, sxsave, sysave;
        volatile int    v __GNU_UNUSED;
        nsave = n;
        xsave = x;
        sxsave = stridex;
        ysave = y;
        sysave = stridey;
        biguns = 0;

        do
        {
LOOP0:
                xsb0 = HI(x);
                hx0 = xsb0 & ~0x80000000;
                if ( hx0 > 0x3fe921fb )
                {
                        biguns = 1;
                        goto MEDIUM;
                }
                if ( hx0 < 0x3e400000 )
                {
```

```
105                         v = *x;
106                         *y = *x;
107                         x += stridex;
108                         y += stridey;
109                         i = 0;
110                         if ( --n <= 0 )
111                                 break;
112                         goto LOOP0;
113                 }
114                 x0 = *x;
115                 py0 = y;
116                 x += stridex;
117                 y += stridey;
118                 i = 1;
119                 if ( --n <= 0 )
120                         break;

122 LOOP1:
123                 xsb1 = HI(x);
124                 hx1 = xsb1 & ~0x80000000;
125                 if ( hx1 > 0x3fe921fb )
126                 {
127                         biguns = 2;
128                         goto MEDIUM;
129                 }
130                 if ( hx1 < 0x3e400000 )
131                 {
132                         v = *x;
133                         *y = *x;
134                         x += stridex;
135                         y += stridey;
136                         i = 1;
137                         if ( --n <= 0 )
138                                 break;
139                         goto LOOP1;
140                 }
141                 x1 = *x;
142                 py1 = y;
143                 x += stridex;
144                 y += stridey;
145                 i = 2;
146                 if ( --n <= 0 )
147                         break;

149 LOOP2:
150                 xsb2 = HI(x);
151                 hx2 = xsb2 & ~0x80000000;
152                 if ( hx2 > 0x3fe921fb )
153                 {
154                         biguns = 3;
155                         goto MEDIUM;
156                 }
157                 if ( hx2 < 0x3e400000 )
158                 {
159                         v = *x;
160                         *y = *x;
161                         x += stridex;
162                         y += stridey;
163                         i = 2;
164                         if ( --n <= 0 )
165                                 break;
166                         goto LOOP2;
167                 }
168                 x2 = *x;
169                 py2 = y;
```



```
171                 i = ( hx0 - 0x3fc90000 ) >> 31;
172                 i |= ( ( hx1 - 0x3fc90000 ) >> 30 ) & 2;
173                 i |= ( ( hx2 - 0x3fc90000 ) >> 29 ) & 4;
174                 switch ( i )
175                 {
176                         double          a0, a1, a2, w0, w1, w2;
177                         double          t0, t1, t2, z0, z1, z2;
178                         unsigned        j0, j1, j2;

180                 case 0:
181                         j0 = ( xsb0 + 0x4000 ) & 0xffff8000;
182                         j1 = ( xsb1 + 0x4000 ) & 0xffff8000;
183                         j2 = ( xsb2 + 0x4000 ) & 0xffff8000;
184                         HI(&t0) = j0;
185                         HI(&t1) = j1;
186                         HI(&t2) = j2;
187                         LO(&t0) = 0;
188                         LO(&t1) = 0;
189                         LO(&t2) = 0;
190                         x0 -= t0;
191                         x1 -= t1;
192                         x2 -= t2;
193                         z0 = x0 * x0;
194                         z1 = x1 * x1;
195                         z2 = x2 * x2;
196                         t0 = z0 * ( qq1 + z0 * qq2 );
197                         t1 = z1 * ( qq1 + z1 * qq2 );
198                         t2 = z2 * ( qq1 + z2 * qq2 );
199                         w0 = x0 * ( one + z0 * ( pp1 + z0 * pp2 ) );
200                         w1 = x1 * ( one + z1 * ( pp1 + z1 * pp2 ) );
201                         w2 = x2 * ( one + z2 * ( pp1 + z2 * pp2 ) );
202                         j0 = ( ( ( j0 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
203                         j1 = ( ( ( j1 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
204                         j2 = ( ( ( j2 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
205                         xsb0 = ( xsb0 >> 30 ) & 2;
206                         xsb1 = ( xsb1 >> 30 ) & 2;
207                         xsb2 = ( xsb2 >> 30 ) & 2;
208                         a0 = __vlibm_TBL_sincos_hi[j0+xsb0];
209                         a1 = __vlibm_TBL_sincos_hi[j1+xsb1];
210                         a2 = __vlibm_TBL_sincos_hi[j2+xsb2];
211                         t0 = ( __vlibm_TBL_sincos_hi[j0+1] * w0 + a0 * t0 ) + __
212                         t1 = ( __vlibm_TBL_sincos_hi[j1+1] * w1 + a1 * t1 ) + __
213                         t2 = ( __vlibm_TBL_sincos_hi[j2+1] * w2 + a2 * t2 ) + __
214                         *py0 = a0 + t0;
215                         *py1 = a1 + t1;
216                         *py2 = a2 + t2;
217                         break;

219                 case 1:
220                         j1 = ( xsb1 + 0x4000 ) & 0xffff8000;
221                         j2 = ( xsb2 + 0x4000 ) & 0xffff8000;
222                         HI(&t1) = j1;
223                         HI(&t2) = j2;
224                         LO(&t1) = 0;
225                         LO(&t2) = 0;
226                         x1 -= t1;
227                         x2 -= t2;
228                         z0 = x0 * x0;
229                         z1 = x1 * x1;
230                         z2 = x2 * x2;
231                         t0 = z0 * ( poly3[0] + z0 * poly4[0] );
232                         t1 = z1 * ( qq1 + z1 * qq2 );
233                         t2 = z2 * ( qq1 + z2 * qq2 );
234                         t0 = z0 * ( poly1[0] + z0 * ( poly2[0] + t0 ) );
235                         w1 = x1 * ( one + z1 * ( pp1 + z1 * pp2 ) );
236                         w2 = x2 * ( one + z2 * ( pp1 + z2 * pp2 ) );
```

```
237                     j1 = ( ( ( j1 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
238                     j2 = ( ( ( j2 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
239                     xsb1 = ( xsb1 >> 30 ) & 2;
240                     xsb2 = ( xsb2 >> 30 ) & 2;
241                     a1 = __vlibm_TBL_sincos_hi[j1+xsb1];
242                     a2 = __vlibm_TBL_sincos_hi[j2+xsb2];
243                     t0 = x0 + x0 * t0;
244                     t1 = ( __vlibm_TBL_sincos_hi[j1+1] * w1 + a1 * t1 ) + __
245                     t2 = ( __vlibm_TBL_sincos_hi[j2+1] * w2 + a2 * t2 ) + __
246                     *py0 = t0;
247                     *py1 = a1 + t1;
248                     *py2 = a2 + t2;
249                     break;

251             case 2:
252                     j0 = ( xsb0 + 0x4000 ) & 0xffff8000;
253                     j2 = ( xsb2 + 0x4000 ) & 0xffff8000;
254                     HI(&t0) = j0;
255                     HI(&t2) = j2;
256                     LO(&t0) = 0;
257                     LO(&t2) = 0;
258                     x0 -= t0;
259                     x2 -= t2;
260                     z0 = x0 * x0;
261                     z1 = x1 * x1;
262                     z2 = x2 * x2;
263                     t0 = z0 * ( qq1 + z0 * qq2 );
264                     t1 = z1 * ( poly3[0] + z1 * poly4[0] );
265                     t2 = z2 * ( qq1 + z2 * qq2 );
266                     w0 = x0 * ( one + z0 * ( pp1 + z0 * pp2 ) );
267                     t1 = z1 * ( poly1[0] + z1 * ( poly2[0] + t1 ) );
268                     w2 = x2 * ( one + z2 * ( pp1 + z2 * pp2 ) );
269                     j0 = ( ( ( j0 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
270                     j2 = ( ( ( j2 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
271                     xsb0 = ( xsb0 >> 30 ) & 2;
272                     xsb2 = ( xsb2 >> 30 ) & 2;
273                     a0 = __vlibm_TBL_sincos_hi[j0+xsb0];
274                     a2 = __vlibm_TBL_sincos_hi[j2+xsb2];
275                     t0 = ( __vlibm_TBL_sincos_hi[j0+1] * w0 + a0 * t0 ) + __
276                     t1 = x1 + x1 * t1;
277                     t2 = ( __vlibm_TBL_sincos_hi[j2+1] * w2 + a2 * t2 ) + __
278                     *py0 = a0 + t0;
279                     *py1 = t1;
280                     *py2 = a2 + t2;
281                     break;

283             case 3:
284                     j2 = ( xsb2 + 0x4000 ) & 0xffff8000;
285                     HI(&t2) = j2;
286                     LO(&t2) = 0;
287                     x2 -= t2;
288                     z0 = x0 * x0;
289                     z1 = x1 * x1;
290                     z2 = x2 * x2;
291                     t0 = z0 * ( poly3[0] + z0 * poly4[0] );
292                     t1 = z1 * ( poly3[0] + z1 * poly4[0] );
293                     t2 = z2 * ( qq1 + z2 * qq2 );
294                     t0 = z0 * ( poly1[0] + z0 * ( poly2[0] + t0 ) );
295                     t1 = z1 * ( poly1[0] + z1 * ( poly2[0] + t1 ) );
296                     w2 = x2 * ( one + z2 * ( pp1 + z2 * pp2 ) );
297                     j2 = ( ( ( j2 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
298                     xsb2 = ( xsb2 >> 30 ) & 2;
299                     a2 = __vlibm_TBL_sincos_hi[j2+xsb2];
300                     t0 = x0 + x0 * t0;
301                     t1 = x1 + x1 * t1;
302                     t2 = ( __vlibm_TBL_sincos_hi[j2+1] * w2 + a2 * t2 ) + __
```



```
303                     *py0 = t0;
304                     *py1 = t1;
305                     *py2 = a2 + t2;
306                     break;

308             case 4:
309                     j0 = ( xsb0 + 0x4000 ) & 0xffff8000;
310                     j1 = ( xsb1 + 0x4000 ) & 0xffff8000;
311                     HI(&t0) = j0;
312                     HI(&t1) = j1;
313                     LO(&t0) = 0;
314                     LO(&t1) = 0;
315                     x0 -= t0;
316                     x1 -= t1;
317                     z0 = x0 * x0;
318                     z1 = x1 * x1;
319                     z2 = x2 * x2;
320                     t0 = z0 * ( qq1 + z0 * qq2 );
321                     t1 = z1 * ( qq1 + z1 * qq2 );
322                     t2 = z2 * ( poly3[0] + z2 * poly4[0] );
323                     w0 = x0 * ( one + z0 * ( pp1 + z0 * pp2 ) );
324                     w1 = x1 * ( one + z1 * ( pp1 + z1 * pp2 ) );
325                     t2 = z2 * ( poly1[0] + z2 * ( poly2[0] + t2 ) );
326                     j0 = ( ( ( j0 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
327                     j1 = ( ( ( j1 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
328                     xsb0 = ( xsb0 >> 30 ) & 2;
329                     xsb1 = ( xsb1 >> 30 ) & 2;
330                     a0 = __vlibm_TBL_sincos_hi[j0+xsb0];
331                     a1 = __vlibm_TBL_sincos_hi[j1+xsb1];
332                     t0 = ( __vlibm_TBL_sincos_hi[j0+1] * w0 + a0 * t0 ) + __
333                     t1 = ( __vlibm_TBL_sincos_hi[j1+1] * w1 + a1 * t1 ) + __
334                     t2 = x2 + x2 * t2;
335                     *py0 = a0 + t0;
336                     *py1 = a1 + t1;
337                     *py2 = t2;
338                     break;

340             case 5:
341                     j1 = ( xsb1 + 0x4000 ) & 0xffff8000;
342                     HI(&t1) = j1;
343                     LO(&t1) = 0;
344                     x1 -= t1;
345                     z0 = x0 * x0;
346                     z1 = x1 * x1;
347                     z2 = x2 * x2;
348                     t0 = z0 * ( poly3[0] + z0 * poly4[0] );
349                     t1 = z1 * ( qq1 + z1 * qq2 );
350                     t2 = z2 * ( poly3[0] + z2 * poly4[0] );
351                     t0 = z0 * ( poly1[0] + z0 * ( poly2[0] + t0 ) );
352                     w1 = x1 * ( one + z1 * ( pp1 + z1 * pp2 ) );
353                     t2 = z2 * ( poly1[0] + z2 * ( poly2[0] + t2 ) );
354                     j1 = ( ( ( j1 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
355                     xsb1 = ( xsb1 >> 30 ) & 2;
356                     a1 = __vlibm_TBL_sincos_hi[j1+xsb1];
357                     t0 = x0 + x0 * t0;
358                     t1 = ( __vlibm_TBL_sincos_hi[j1+1] * w1 + a1 * t1 ) + __
359                     t2 = x2 + x2 * t2;
360                     *py0 = t0;
361                     *py1 = a1 + t1;
362                     *py2 = t2;
363                     break;

365             case 6:
366                     j0 = ( xsb0 + 0x4000 ) & 0xffff8000;
367                     HI(&t0) = j0;
368                     LO(&t0) = 0;
```

```
369                     x0 -= t0;
370                     z0 = x0 * x0;
371                     z1 = x1 * x1;
372                     z2 = x2 * x2;
373                     t0 = z0 * ( qq1 + z0 * qq2 );
374                     t1 = z1 * ( poly3[0] + z1 * poly4[0] );
375                     t2 = z2 * ( poly3[0] + z2 * poly4[0] );
376                     w0 = x0 * ( one + z0 * ( pp1 + z0 * pp2 ) );
377                     t1 = z1 * ( poly1[0] + z1 * ( poly2[0] + t1 ) );
378                     t2 = z2 * ( poly1[0] + z2 * ( poly2[0] + t2 ) );
379                     j0 = ( ( ( j0 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
380                     xsb0 = ( xsb0 >> 30 ) & 2;
381                     a0 = __vlibm_TBL_sincos_hi[j0+xsb0];
382                     t0 = ( __vlibm_TBL_sincos_hi[j0+1] * w0 + a0 * t0 ) + __
383                     t1 = x1 + x1 * t1;
384                     t2 = x2 + x2 * t2;
385                     *py0 = a0 + t0;
386                     *py1 = t1;
387                     *py2 = t2;
388                     break;

390             case 7:
391                     z0 = x0 * x0;
392                     z1 = x1 * x1;
393                     z2 = x2 * x2;
394                     t0 = z0 * ( poly3[0] + z0 * poly4[0] );
395                     t1 = z1 * ( poly3[0] + z1 * poly4[0] );
396                     t2 = z2 * ( poly3[0] + z2 * poly4[0] );
397                     t0 = z0 * ( poly1[0] + z0 * ( poly2[0] + t0 ) );
398                     t1 = z1 * ( poly1[0] + z1 * ( poly2[0] + t1 ) );
399                     t2 = z2 * ( poly1[0] + z2 * ( poly2[0] + t2 ) );
400                     t0 = x0 + x0 * t0;
401                     t1 = x1 + x1 * t1;
402                     t2 = x2 + x2 * t2;
403                     *py0 = t0;
404                     *py1 = t1;
405                     *py2 = t2;
406                     break;
407             }

409             x += stridex;
410             y += stridey;
411             i = 0;
412     } while ( --n > 0 );

414     if ( i > 0 )
415     {
416             double          a0, a1, w0, w1;
417             double          t0, t1, z0, z1;
418             unsigned        j0, j1;

420             if ( i > 1 )
421             {
422                     if ( hx1 < 0x3fc90000 )
423                     {
424                             z1 = x1 * x1;
425                             t1 = z1 * ( poly3[0] + z1 * poly4[0] );
426                             t1 = z1 * ( poly1[0] + z1 * ( poly2[0] + t1 ) );
427                             t1 = x1 + x1 * t1;
428                             *py1 = t1;
429                     }
430                     else
431                     {
432                             j1 = ( xsb1 + 0x4000 ) & 0xffff8000;
433                             HI(&t1) = j1;
434                             LO(&t1) = 0;
```



```
435                             x1 -= t1;
436                             z1 = x1 * x1;
437                             t1 = z1 * ( qq1 + z1 * qq2 );
438                             w1 = x1 * ( one + z1 * ( pp1 + z1 * pp2 ) );
439                             j1 = ( ( ( j1 & ~0x80000000 ) - 0x3fc40000 ) >>
440                             xsb1 = ( xsb1 >> 30 ) & 2;
441                             a1 = __vlibm_TBL_sincos_hi[j1+xsb1];
442                             t1 = ( __vlibm_TBL_sincos_hi[j1+1] * w1 + a1 * t
443                             *py1 = a1 + t1;
444                     }
445             }
446             if ( hx0 < 0x3fc90000 )
447             {
448                     z0 = x0 * x0;
449                     t0 = z0 * ( poly3[0] + z0 * poly4[0] );
450                     t0 = z0 * ( poly1[0] + z0 * ( poly2[0] + t0 ) );
451                     t0 = x0 + x0 * t0;
452                     *py0 = t0;
453             }
454             else
455             {
456                     j0 = ( xsb0 + 0x4000 ) & 0xffff8000;
457                     HI(&t0) = j0;
458                     LO(&t0) = 0;
459                     x0 -= t0;
460                     z0 = x0 * x0;
461                     t0 = z0 * ( qq1 + z0 * qq2 );
462                     w0 = x0 * ( one + z0 * ( pp1 + z0 * pp2 ) );
463                     j0 = ( ( ( j0 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
464                     xsb0 = ( xsb0 >> 30 ) & 2;
465                     a0 = __vlibm_TBL_sincos_hi[j0+xsb0];
466                     t0 = ( __vlibm_TBL_sincos_hi[j0+1] * w0 + a0 * t0 ) + __
467                     *py0 = a0 + t0;
468             }
469     }

471     return;

473     /*
474      * MEDIUM RANGE PROCESSING
475      * Jump here at first sign of medium range argument.  We are a bit
476      * confused due to the jump.. fix up several variables and jump into
477      * the nth loop, same as was being processed above.
478      */

480 MEDIUM:

482     x0_or_one[1] = 1.0;
483     x1_or_one[1] = 1.0;
484     x2_or_one[1] = 1.0;
485     x0_or_one[3] = -1.0;
486     x1_or_one[3] = -1.0;
487     x2_or_one[3] = -1.0;
488     y0_or_zero[1] = 0.0;
489     y1_or_zero[1] = 0.0;
490     y2_or_zero[1] = 0.0;
491     y0_or_zero[3] = 0.0;
492     y1_or_zero[3] = 0.0;
493     y2_or_zero[3] = 0.0;

495     if ( biguns == 3 )
496     {
497             biguns = 0;
498             xsb0 = xsb0 >> 31;
499             xsb1 = xsb1 >> 31;
500             goto loop2;
```

```
501         }
502         else if ( biguns == 2 )
503         {
504                 xsb0 = xsb0 >> 31;
505                 biguns = 0;
506                 goto loop1;
507         }
508         biguns = 0;

510         do
511         {
512                 double          fn0, fn1, fn2, a0, a1, a2, w0, w1, w2, y0, y1, y
513                 unsigned        hx;
514                 int             n0, n1, n2;

516 loop0:
517                 hx = HI(x);
518                 xsb0 = hx >> 31;
519                 hx &= ~0x80000000;
520                 if ( hx < 0x3e400000 )
521                 {
522                         v = *x;
523                         *y = *x;
524                         x += stridex;
525                         y += stridey;
526                         i = 0;
527                         if ( --n <= 0 )
528                                 break;
529                         goto loop0;
530                 }
531                 if ( hx > 0x413921fb )
532                 {
533                         if ( hx >= 0x7ff00000 )
534                         {
535                                 x0 = *x;
536                                 *y = x0 - x0;
537                         }
538                         else
539                                 biguns = 1;
540                         x += stridex;
541                         y += stridey;
542                         i = 0;
543                         if ( --n <= 0 )
544                                 break;
545                         goto loop0;
546                 }
547                 x0 = *x;
548                 py0 = y;
549                 x += stridex;
550                 y += stridey;
551                 i = 1;
552                 if ( --n <= 0 )
553                         break;

555 loop1:
556                 hx = HI(x);
557                 xsb1 = hx >> 31;
558                 hx &= ~0x80000000;
559                 if ( hx < 0x3e400000 )
560                 {
561                         v = *x;
562                         *y = *x;
563                         x += stridex;
564                         y += stridey;
565                         i = 1;
566                         if ( --n <= 0 )
```



```
567                                 break;
568                         goto loop1;
569                 }
570                 if ( hx > 0x413921fb )
571                 {
572                         if ( hx >= 0x7ff00000 )
573                         {
574                                 x1 = *x;
575                                 *y = x1 - x1;
576                         }
577                         else
578                                 biguns = 1;
579                         x += stridex;
580                         y += stridey;
581                         i = 1;
582                         if ( --n <= 0 )
583                                 break;
584                         goto loop1;
585                 }
586                 x1 = *x;
587                 py1 = y;
588                 x += stridex;
589                 y += stridey;
590                 i = 2;
591                 if ( --n <= 0 )
592                         break;

594 loop2:
595                 hx = HI(x);
596                 xsb2 = hx >> 31;
597                 hx &= ~0x80000000;
598                 if ( hx < 0x3e400000 )
599                 {
600                         v = *x;
601                         *y = *x;
602                         x += stridex;
603                         y += stridey;
604                         i = 2;
605                         if ( --n <= 0 )
606                                 break;
607                         goto loop2;
608                 }
609                 if ( hx > 0x413921fb )
610                 {
611                         if ( hx >= 0x7ff00000 )
612                         {
613                                 x2 = *x;
614                                 *y = x2 - x2;
615                         }
616                         else
617                                 biguns = 1;
618                         x += stridex;
619                         y += stridey;
620                         i = 2;
621                         if ( --n <= 0 )
622                                 break;
623                         goto loop2;
624                 }
625                 x2 = *x;
626                 py2 = y;

628                 n0 = (int) ( x0 * invpio2 + half[xsb0] );
629                 n1 = (int) ( x1 * invpio2 + half[xsb1] );
630                 n2 = (int) ( x2 * invpio2 + half[xsb2] );
631                 fn0 = (double) n0;
632                 fn1 = (double) n1;
```

```
633                 fn2 = (double) n2;
634                 n0 &= 3;
635                 n1 &= 3;
636                 n2 &= 3;
637                 a0 = x0 - fn0 * pio2_1;
638                 a1 = x1 - fn1 * pio2_1;
639                 a2 = x2 - fn2 * pio2_1;
640                 w0 = fn0 * pio2_2;
641                 w1 = fn1 * pio2_2;
642                 w2 = fn2 * pio2_2;
643                 x0 = a0 - w0;
644                 x1 = a1 - w1;
645                 x2 = a2 - w2;
646                 y0 = ( a0 - x0 ) - w0;
647                 y1 = ( a1 - x1 ) - w1;
648                 y2 = ( a2 - x2 ) - w2;
649                 a0 = x0;
650                 a1 = x1;
651                 a2 = x2;
652                 w0 = fn0 * pio2_3 - y0;
653                 w1 = fn1 * pio2_3 - y1;
654                 w2 = fn2 * pio2_3 - y2;
655                 x0 = a0 - w0;
656                 x1 = a1 - w1;
657                 x2 = a2 - w2;
658                 y0 = ( a0 - x0 ) - w0;
659                 y1 = ( a1 - x1 ) - w1;
660                 y2 = ( a2 - x2 ) - w2;
661                 a0 = x0;
662                 a1 = x1;
663                 a2 = x2;
664                 w0 = fn0 * pio2_3t - y0;
665                 w1 = fn1 * pio2_3t - y1;
666                 w2 = fn2 * pio2_3t - y2;
667                 x0 = a0 - w0;
668                 x1 = a1 - w1;
669                 x2 = a2 - w2;
670                 y0 = ( a0 - x0 ) - w0;
671                 y1 = ( a1 - x1 ) - w1;
672                 y2 = ( a2 - x2 ) - w2;
673                 xsb0 = HI(&x0);
674                 i = ( ( xsb0 & ~0x80000000 ) - thresh[n0&1] ) >> 31;
675                 xsb1 = HI(&x1);
676                 i |= ( ( ( xsb1 & ~0x80000000 ) - thresh[n1&1] ) >> 30 ) & 2;
677                 xsb2 = HI(&x2);
678                 i |= ( ( ( xsb2 & ~0x80000000 ) - thresh[n2&1] ) >> 29 ) & 4;
679                 switch ( i )
680                 {
681                         double          t0, t1, t2, z0, z1, z2;
682                         unsigned        j0, j1, j2;
683
684                 case 0:
685                         j0 = ( xsb0 + 0x4000 ) & 0xffff8000;
686                         j1 = ( xsb1 + 0x4000 ) & 0xffff8000;
687                         j2 = ( xsb2 + 0x4000 ) & 0xffff8000;
688                         HI(&t0) = j0;
689                         HI(&t1) = j1;
690                         HI(&t2) = j2;
691                         LO(&t0) = 0;
692                         LO(&t1) = 0;
693                         LO(&t2) = 0;
694                         x0 = ( x0 - t0 ) + y0;
695                         x1 = ( x1 - t1 ) + y1;
696                         x2 = ( x2 - t2 ) + y2;
697                         z0 = x0 * x0;
698                         z1 = x1 * x1;
```



```
699                         z2 = x2 * x2;
700                         t0 = z0 * ( qq1 + z0 * qq2 );
701                         t1 = z1 * ( qq1 + z1 * qq2 );
702                         t2 = z2 * ( qq1 + z2 * qq2 );
703                         w0 = x0 * ( one + z0 * ( pp1 + z0 * pp2 ) );
704                         w1 = x1 * ( one + z1 * ( pp1 + z1 * pp2 ) );
705                         w2 = x2 * ( one + z2 * ( pp1 + z2 * pp2 ) );
706                         j0 = ( ( ( j0 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
707                         j1 = ( ( ( j1 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
708                         j2 = ( ( ( j2 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
709                         xsb0 = ( xsb0 >> 30 ) & 2;
710                         xsb1 = ( xsb1 >> 30 ) & 2;
711                         xsb2 = ( xsb2 >> 30 ) & 2;
712                         n0 ^= ( xsb0 & ~( n0 << 1 ) );
713                         n1 ^= ( xsb1 & ~( n1 << 1 ) );
714                         n2 ^= ( xsb2 & ~( n2 << 1 ) );
715                         xsb0 |= 1;
716                         xsb1 |= 1;
717                         xsb2 |= 1;
718                         a0 = __vlibm_TBL_sincos_hi[j0+n0];
719                         a1 = __vlibm_TBL_sincos_hi[j1+n1];
720                         a2 = __vlibm_TBL_sincos_hi[j2+n2];
721                         t0 = ( __vlibm_TBL_sincos_hi[j0+((n0+xsb0)&3)] * w0 + a0
722                         t1 = ( __vlibm_TBL_sincos_hi[j1+((n1+xsb1)&3)] * w1 + a1
723                         t2 = ( __vlibm_TBL_sincos_hi[j2+((n2+xsb2)&3)] * w2 + a2
724                         *py0 = ( a0 + t0 );
725                         *py1 = ( a1 + t1 );
726                         *py2 = ( a2 + t2 );
727                         break;
728
729                 case 1:
730                         j0 = n0 & 1;
731                         j1 = ( xsb1 + 0x4000 ) & 0xffff8000;
732                         j2 = ( xsb2 + 0x4000 ) & 0xffff8000;
733                         HI(&t1) = j1;
734                         HI(&t2) = j2;
735                         LO(&t1) = 0;
736                         LO(&t2) = 0;
737                         x0_or_one[0] = x0;
738                         x0_or_one[2] = -x0;
739                         y0_or_zero[0] = y0;
740                         y0_or_zero[2] = -y0;
741                         x1 = ( x1 - t1 ) + y1;
742                         x2 = ( x2 - t2 ) + y2;
743                         z0 = x0 * x0;
744                         z1 = x1 * x1;
745                         z2 = x2 * x2;
746                         t0 = z0 * ( poly3[j0] + z0 * poly4[j0] );
747                         t1 = z1 * ( qq1 + z1 * qq2 );
748                         t2 = z2 * ( qq1 + z2 * qq2 );
749                         t0 = z0 * ( poly1[j0] + z0 * ( poly2[j0] + t0 ) );
750                         w1 = x1 * ( one + z1 * ( pp1 + z1 * pp2 ) );
751                         w2 = x2 * ( one + z2 * ( pp1 + z2 * pp2 ) );
752                         j1 = ( ( ( j1 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
753                         j2 = ( ( ( j2 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
754                         xsb1 = ( xsb1 >> 30 ) & 2;
755                         xsb2 = ( xsb2 >> 30 ) & 2;
756                         n1 ^= ( xsb1 & ~( n1 << 1 ) );
757                         n2 ^= ( xsb2 & ~( n2 << 1 ) );
758                         xsb1 |= 1;
759                         xsb2 |= 1;
760                         a1 = __vlibm_TBL_sincos_hi[j1+n1];
761                         a2 = __vlibm_TBL_sincos_hi[j2+n2];
762                         t0 = x0_or_one[n0] + ( y0_or_zero[n0] + x0_or_one[n0] *
763                         t1 = ( __vlibm_TBL_sincos_hi[j1+((n1+xsb1)&3)] * w1 + a1
764                         t2 = ( __vlibm_TBL_sincos_hi[j2+((n2+xsb2)&3)] * w2 + a2
```

```
765                         *py0 = t0;
766                         *py1 = ( a1 + t1 );
767                         *py2 = ( a2 + t2 );
768                         break;

770                 case 2:
771                         j0 = ( xsb0 + 0x4000 ) & 0xffff8000;
772                         j1 = n1 & 1;
773                         j2 = ( xsb2 + 0x4000 ) & 0xffff8000;
774                         HI(&t0) = j0;
775                         HI(&t2) = j2;
776                         LO(&t0) = 0;
777                         LO(&t2) = 0;
778                         x1_or_one[0] = x1;
779                         x1_or_one[2] = -x1;
780                         x0 = ( x0 - t0 ) + y0;
781                         y1_or_zero[0] = y1;
782                         y1_or_zero[2] = -y1;
783                         x2 = ( x2 - t2 ) + y2;
784                         z0 = x0 * x0;
785                         z1 = x1 * x1;
786                         z2 = x2 * x2;
787                         t0 = z0 * ( qq1 + z0 * qq2 );
788                         t1 = z1 * ( poly3[j1] + z1 * poly4[j1] );
789                         t2 = z2 * ( qq1 + z2 * qq2 );
790                         w0 = x0 * ( one + z0 * ( pp1 + z0 * pp2 ) );
791                         t1 = z1 * ( poly1[j1] + z1 * ( poly2[j1] + t1 ) );
792                         w2 = x2 * ( one + z2 * ( pp1 + z2 * pp2 ) );
793                         j0 = ( ( ( j0 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
794                         j2 = ( ( ( j2 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
795                         xsb0 = ( xsb0 >> 30 ) & 2;
796                         xsb2 = ( xsb2 >> 30 ) & 2;
797                         n0 ^= ( xsb0 & ~( n0 << 1 ) );
798                         n2 ^= ( xsb2 & ~( n2 << 1 ) );
799                         xsb0 |= 1;
800                         xsb2 |= 1;
801                         a0 = __vlibm_TBL_sincos_hi[j0+n0];
802                         a2 = __vlibm_TBL_sincos_hi[j2+n2];
803                         t0 = ( __vlibm_TBL_sincos_hi[j0+((n0+xsb0)&3)] * w0 + a0
804                         t1 = x1_or_one[n1] + ( y1_or_zero[n1] + x1_or_one[n1] *
805                         t2 = ( __vlibm_TBL_sincos_hi[j2+((n2+xsb2)&3)] * w2 + a2
806                         *py0 = ( a0 + t0 );
807                         *py1 = t1;
808                         *py2 = ( a2 + t2 );
809                         break;

811                 case 3:
812                         j0 = n0 & 1;
813                         j1 = n1 & 1;
814                         j2 = ( xsb2 + 0x4000 ) & 0xffff8000;
815                         HI(&t2) = j2;
816                         LO(&t2) = 0;
817                         x0_or_one[0] = x0;
818                         x0_or_one[2] = -x0;
819                         x1_or_one[0] = x1;
820                         x1_or_one[2] = -x1;
821                         y0_or_zero[0] = y0;
822                         y0_or_zero[2] = -y0;
823                         y1_or_zero[0] = y1;
824                         y1_or_zero[2] = -y1;
825                         x2 = ( x2 - t2 ) + y2;
826                         z0 = x0 * x0;
827                         z1 = x1 * x1;
828                         z2 = x2 * x2;
829                         t0 = z0 * ( poly3[j0] + z0 * poly4[j0] );
830                         t1 = z1 * ( poly3[j1] + z1 * poly4[j1] );
```



```
831                         t2 = z2 * ( qq1 + z2 * qq2 );
832                         t0 = z0 * ( poly1[j0] + z0 * ( poly2[j0] + t0 ) );
833                         t1 = z1 * ( poly1[j1] + z1 * ( poly2[j1] + t1 ) );
834                         w2 = x2 * ( one + z2 * ( pp1 + z2 * pp2 ) );
835                         j2 = ( ( ( j2 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
836                         xsb2 = ( xsb2 >> 30 ) & 2;
837                         n2 ^= ( xsb2 & ~( n2 << 1 ) );
838                         xsb2 |= 1;
839                         a2 = __vlibm_TBL_sincos_hi[j2+n2];
840                         t0 = x0_or_one[n0] + ( y0_or_zero[n0] + x0_or_one[n0] *
841                         t1 = x1_or_one[n1] + ( y1_or_zero[n1] + x1_or_one[n1] *
842                         t2 = ( __vlibm_TBL_sincos_hi[j2+((n2+xsb2)&3)] * w2 + a2
843                         *py0 = t0;
844                         *py1 = t1;
845                         *py2 = ( a2 + t2 );
846                         break;

848                 case 4:
849                         j0 = ( xsb0 + 0x4000 ) & 0xffff8000;
850                         j1 = ( xsb1 + 0x4000 ) & 0xffff8000;
851                         j2 = n2 & 1;
852                         HI(&t0) = j0;
853                         HI(&t1) = j1;
854                         LO(&t0) = 0;
855                         LO(&t1) = 0;
856                         x2_or_one[0] = x2;
857                         x2_or_one[2] = -x2;
858                         x0 = ( x0 - t0 ) + y0;
859                         x1 = ( x1 - t1 ) + y1;
860                         y2_or_zero[0] = y2;
861                         y2_or_zero[2] = -y2;
862                         z0 = x0 * x0;
863                         z1 = x1 * x1;
864                         z2 = x2 * x2;
865                         t0 = z0 * ( qq1 + z0 * qq2 );
866                         t1 = z1 * ( qq1 + z1 * qq2 );
867                         t2 = z2 * ( poly3[j2] + z2 * poly4[j2] );
868                         w0 = x0 * ( one + z0 * ( pp1 + z0 * pp2 ) );
869                         w1 = x1 * ( one + z1 * ( pp1 + z1 * pp2 ) );
870                         t2 = z2 * ( poly1[j2] + z2 * ( poly2[j2] + t2 ) );
871                         j0 = ( ( ( j0 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
872                         j1 = ( ( ( j1 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
873                         xsb0 = ( xsb0 >> 30 ) & 2;
874                         xsb1 = ( xsb1 >> 30 ) & 2;
875                         n0 ^= ( xsb0 & ~( n0 << 1 ) );
876                         n1 ^= ( xsb1 & ~( n1 << 1 ) );
877                         xsb0 |= 1;
878                         xsb1 |= 1;
879                         a0 = __vlibm_TBL_sincos_hi[j0+n0];
880                         a1 = __vlibm_TBL_sincos_hi[j1+n1];
881                         t0 = ( __vlibm_TBL_sincos_hi[j0+((n0+xsb0)&3)] * w0 + a0
882                         t1 = ( __vlibm_TBL_sincos_hi[j1+((n1+xsb1)&3)] * w1 + a1
883                         t2 = x2_or_one[n2] + ( y2_or_zero[n2] + x2_or_one[n2] *
884                         *py0 = ( a0 + t0 );
885                         *py1 = ( a1 + t1 );
886                         *py2 = t2;
887                         break;

889                 case 5:
890                         j0 = n0 & 1;
891                         j1 = ( xsb1 + 0x4000 ) & 0xffff8000;
892                         j2 = n2 & 1;
893                         HI(&t1) = j1;
894                         LO(&t1) = 0;
895                         x0_or_one[0] = x0;
896                         x0_or_one[2] = -x0;
```

```
897                         x2_or_one[0] = x2;
898                         x2_or_one[2] = -x2;
899                         y0_or_zero[0] = y0;
900                         y0_or_zero[2] = -y0;
901                         x1 = ( x1 - t1 ) + y1;
902                         y2_or_zero[0] = y2;
903                         y2_or_zero[2] = -y2;
904                         z0 = x0 * x0;
905                         z1 = x1 * x1;
906                         z2 = x2 * x2;
907                         t0 = z0 * ( poly3[j0] + z0 * poly4[j0] );
908                         t1 = z1 * ( qq1 + z1 * qq2 );
909                         t2 = z2 * ( poly3[j2] + z2 * poly4[j2] );
910                         t0 = z0 * ( poly1[j0] + z0 * ( poly2[j0] + t0 ) );
911                         w1 = x1 * ( one + z1 * ( pp1 + z1 * pp2 ) );
912                         t2 = z2 * ( poly1[j2] + z2 * ( poly2[j2] + t2 ) );
913                         j1 = ( ( ( j1 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
914                         xsb1 = ( xsb1 >> 30 ) & 2;
915                         n1 ^= ( xsb1 & ~( n1 << 1 ) );
916                         xsb1 |= 1;
917                         a1 = __vlibm_TBL_sincos_hi[j1+n1];
918                         t0 = x0_or_one[n0] + ( y0_or_zero[n0] + x0_or_one[n0] *
919                         t1 = ( __vlibm_TBL_sincos_hi[j1+((n1+xsb1)&3)] * w1 + a1
920                         t2 = x2_or_one[n2] + ( y2_or_zero[n2] + x2_or_one[n2] *
921                         *py0 = t0;
922                         *py1 = ( a1 + t1 );
923                         *py2 = t2;
924                         break;

926                 case 6:
927                         j0 = ( xsb0 + 0x4000 ) & 0xffff8000;
928                         j1 = n1 & 1;
929                         j2 = n2 & 1;
930                         HI(&t0) = j0;
931                         LO(&t0) = 0;
932                         x1_or_one[0] = x1;
933                         x1_or_one[2] = -x1;
934                         x2_or_one[0] = x2;
935                         x2_or_one[2] = -x2;
936                         x0 = ( x0 - t0 ) + y0;
937                         y1_or_zero[0] = y1;
938                         y1_or_zero[2] = -y1;
939                         y2_or_zero[0] = y2;
940                         y2_or_zero[2] = -y2;
941                         z0 = x0 * x0;
942                         z1 = x1 * x1;
943                         z2 = x2 * x2;
944                         t0 = z0 * ( qq1 + z0 * qq2 );
945                         t1 = z1 * ( poly3[j1] + z1 * poly4[j1] );
946                         t2 = z2 * ( poly3[j2] + z2 * poly4[j2] );
947                         w0 = x0 * ( one + z0 * ( pp1 + z0 * pp2 ) );
948                         t1 = z1 * ( poly1[j1] + z1 * ( poly2[j1] + t1 ) );
949                         t2 = z2 * ( poly1[j2] + z2 * ( poly2[j2] + t2 ) );
950                         j0 = ( ( ( j0 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
951                         xsb0 = ( xsb0 >> 30 ) & 2;
952                         n0 ^= ( xsb0 & ~( n0 << 1 ) );
953                         xsb0 |= 1;
954                         a0 = __vlibm_TBL_sincos_hi[j0+n0];
955                         t0 = ( __vlibm_TBL_sincos_hi[j0+((n0+xsb0)&3)] * w0 + a0
956                         t1 = x1_or_one[n1] + ( y1_or_zero[n1] + x1_or_one[n1] *
957                         t2 = x2_or_one[n2] + ( y2_or_zero[n2] + x2_or_one[n2] *
958                         *py0 = ( a0 + t0 );
959                         *py1 = t1;
960                         *py2 = t2;
961                         break;
```



```
963                 case 7:
964                         j0 = n0 & 1;
965                         j1 = n1 & 1;
966                         j2 = n2 & 1;
967                         x0_or_one[0] = x0;
968                         x0_or_one[2] = -x0;
969                         x1_or_one[0] = x1;
970                         x1_or_one[2] = -x1;
971                         x2_or_one[0] = x2;
972                         x2_or_one[2] = -x2;
973                         y0_or_zero[0] = y0;
974                         y0_or_zero[2] = -y0;
975                         y1_or_zero[0] = y1;
976                         y1_or_zero[2] = -y1;
977                         y2_or_zero[0] = y2;
978                         y2_or_zero[2] = -y2;
979                         z0 = x0 * x0;
980                         z1 = x1 * x1;
981                         z2 = x2 * x2;
982                         t0 = z0 * ( poly3[j0] + z0 * poly4[j0] );
983                         t1 = z1 * ( poly3[j1] + z1 * poly4[j1] );
984                         t2 = z2 * ( poly3[j2] + z2 * poly4[j2] );
985                         t0 = z0 * ( poly1[j0] + z0 * ( poly2[j0] + t0 ) );
986                         t1 = z1 * ( poly1[j1] + z1 * ( poly2[j1] + t1 ) );
987                         t2 = z2 * ( poly1[j2] + z2 * ( poly2[j2] + t2 ) );
988                         t0 = x0_or_one[n0] + ( y0_or_zero[n0] + x0_or_one[n0] *
989                         t1 = x1_or_one[n1] + ( y1_or_zero[n1] + x1_or_one[n1] *
990                         t2 = x2_or_one[n2] + ( y2_or_zero[n2] + x2_or_one[n2] *
991                         *py0 = t0;
992                         *py1 = t1;
993                         *py2 = t2;
994                         break;
995                 }

997                 x += stridex;
998                 y += stridey;
999                 i = 0;
1000        } while ( --n > 0 );

1002        if ( i > 0 )
1003        {
1004                double          fn0, fn1, a0, a1, w0, w1, y0, y1;
1005                double          t0, t1, z0, z1;
1006                unsigned        j0, j1;
1007                int             n0, n1;

1009                if ( i > 1 )
1010                {
1011                        n1 = (int) ( x1 * invpio2 + half[xsb1] );
1012                        fn1 = (double) n1;
1013                        n1 &= 3;
1014                        a1 = x1 - fn1 * pio2_1;
1015                        w1 = fn1 * pio2_2;
1016                        x1 = a1 - w1;
1017                        y1 = ( a1 - x1 ) - w1;
1018                        a1 = x1;
1019                        w1 = fn1 * pio2_3 - y1;
1020                        x1 = a1 - w1;
1021                        y1 = ( a1 - x1 ) - w1;
1022                        a1 = x1;
1023                        w1 = fn1 * pio2_3t - y1;
1024                        x1 = a1 - w1;
1025                        y1 = ( a1 - x1 ) - w1;
1026                        xsb1 = HI(&x1);
1027                        if ( ( xsb1 & ~0x80000000 ) < thresh[n1&1] )
1028                        {
```

```
1029                                j1 = n1 & 1;
1030                                x1_or_one[0] = x1;
1031                                x1_or_one[2] = -x1;
1032                                y1_or_zero[0] = y1;
1033                                y1_or_zero[2] = -y1;
1034                                z1 = x1 * x1;
1035                                t1 = z1 * ( poly3[j1] + z1 * poly4[j1] );
1036                                t1 = z1 * ( poly1[j1] + z1 * ( poly2[j1] + t1 )
1037                                t1 = x1_or_one[n1] + ( y1_or_zero[n1] + x1_or_on
1038                                *py1 = t1;
1039                        }
1040                        else
1041                        {
1042                                j1 = ( xsb1 + 0x4000 ) & 0xffff8000;
1043                                HI(&t1) = j1;
1044                                LO(&t1) = 0;
1045                                x1 = ( x1 - t1 ) + y1;
1046                                z1 = x1 * x1;
1047                                t1 = z1 * ( qq1 + z1 * qq2 );
1048                                w1 = x1 * ( one + z1 * ( pp1 + z1 * pp2 ) );
1049                                j1 = ( ( ( j1 & ~0x80000000 ) - 0x3fc40000 ) >>
1050                                xsb1 = ( xsb1 >> 30 ) & 2;
1051                                n1 ^= ( xsb1 & ~( n1 << 1 ) );
1052                                xsb1 |= 1;
1053                                a1 = __vlibm_TBL_sincos_hi[j1+n1];
1054                                t1 = ( __vlibm_TBL_sincos_hi[j1+((n1+xsb1)&3)] *
1055                                *py1 = ( a1 + t1 );
1056                        }
1057                }
1058                n0 = (int) ( x0 * invpio2 + half[xsb0] );
1059                fn0 = (double) n0;
1060                n0 &= 3;
1061                a0 = x0 - fn0 * pio2_1;
1062                w0 = fn0 * pio2_2;
1063                x0 = a0 - w0;
1064                y0 = ( a0 - x0 ) - w0;
1065                a0 = x0;
1066                w0 = fn0 * pio2_3 - y0;
1067                x0 = a0 - w0;
1068                y0 = ( a0 - x0 ) - w0;
1069                a0 = x0;
1070                w0 = fn0 * pio2_3t - y0;
1071                x0 = a0 - w0;
1072                y0 = ( a0 - x0 ) - w0;
1073                xsb0 = HI(&x0);
1074                if ( ( xsb0 & ~0x80000000 ) < thresh[n0&1] )
1075                {
1076                        j0 = n0 & 1;
1077                        x0_or_one[0] = x0;
1078                        x0_or_one[2] = -x0;
1079                        y0_or_zero[0] = y0;
1080                        y0_or_zero[2] = -y0;
1081                        z0 = x0 * x0;
1082                        t0 = z0 * ( poly3[j0] + z0 * poly4[j0] );
1083                        t0 = z0 * ( poly1[j0] + z0 * ( poly2[j0] + t0 ) );
1084                        t0 = x0_or_one[n0] + ( y0_or_zero[n0] + x0_or_one[n0] *
1085                        *py0 = t0;
1086                }
1087                else
1088                {
1089                        j0 = ( xsb0 + 0x4000 ) & 0xffff8000;
1090                        HI(&t0) = j0;
1091                        LO(&t0) = 0;
1092                        x0 = ( x0 - t0 ) + y0;
1093                        z0 = x0 * x0;
1094                        t0 = z0 * ( qq1 + z0 * qq2 );
```



```
1095                        w0 = x0 * ( one + z0 * ( pp1 + z0 * pp2 ) );
1096                        j0 = ( ( ( j0 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
1097                        xsb0 = ( xsb0 >> 30 ) & 2;
1098                        n0 ^= ( xsb0 & ~( n0 << 1 ) );
1099                        xsb0 |= 1;
1100                        a0 = __vlibm_TBL_sincos_hi[j0+n0];
1101                        t0 = ( __vlibm_TBL_sincos_hi[j0+((n0+xsb0)&3)] * w0 + a0
1102                        *py0 = ( a0 + t0 );
1103                }
1104        }

1106        if ( biguns )
1107                __vlibm_vsin_big( nsave, xsave, sxsave, ysave, sysave, 0x413921f
1108 }
```

```
**********************************************************
   39440 Thu Oct  9 19:48:55 2014
new/usr/src/lib/libmvec/common/__vsincos.c
Revert "remove unused v from libmvec"
This reverts commit e853d278ee4b7f2a8c2117cc598cfc68b4e3f29b.
fix system-library-math.mf
update libm manifests
14071:dece9aafe99a - fix build problems on sparc
remove unused v from libmvec
fix for patch09 - use __GNU_UNUSED
patch12 - math.h: Align things with GCC
patch11 - added LIBM man pages
patch09 - update libmvec: fix build issues by gcc46
patch08 - libmvec: fixed compilation issues after updates
patch01 - 693 import Sun Devpro Math Library
Revert "remove unused v from libmvec"
This reverts commit e853d278ee4b7f2a8c2117cc598cfc68b4e3f29b.
fix system-library-math.mf
update libm manifests
14071:dece9aafe99a - fix build problems on sparc
remove unused v from libmvec
fix for patch09 - use __GNU_UNUSED
patch12 - math.h: Align things with GCC
patch11 - added LIBM man pages
patch09 - update libmvec: fix build issues by gcc46
patch08 - libmvec: fixed compilation issues after updates
patch01 - 693 import Sun Devpro Math Library
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" 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.
 */

#include <sys/isa_defs.h>
#include <sys/ccompile.h>

#ifdef _LITTLE_ENDIAN
#define HI(x)   *(1+(int*)x)
#define LO(x)   *(unsigned*)x
#else
#define HI(x)   *(int*)x
#define LO(x)   *(1+(unsigned*)x)
```



```
#endif

#ifdef __RESTRICT
#define restrict _Restrict
#else
#define restrict
#endif

/*
 * vsincos.c
 *
 * Vector sine and cosine function.  Just slight modifications to vcos.c.
 */

extern const double __vlibm_TBL_sincos_hi[], __vlibm_TBL_sincos_lo[];

static const double
        half[2] = { 0.5, -0.5 },
        one             = 1.0,
        invpio2 = 0.6366197723675813430755350,  /* 53 bits of pi/2 */
        pio2_1  = 1.57079632673412561416600,  /* first 33 bits of pi/2 */
        pio2_2  = 6.07710050630396597659600e-11, /* second 33 bits of pi/2 */
        pio2_3  = 2.02226248711166455796000e-21, /* third 33 bits of pi/2 */
        pio2_3t = 8.47842766036889964395900e-32, /* pi/2 - pio2_3 */
        pp1             = -1.66666666660576046527626394313498255467600e-0001,
        pp2             =  8.33326120969096312671837656614618094444200e-0003,
        qq1             = -4.99999999977710986407023955908711557870000e-0001,
        qq2             =  4.16665486385721935064505588101884208958000e-0002,
        poly1[2]= { -1.66666666666629669805215138920301589656000e-0001,
                                -4.99999999999931701464060878888294524481000e-0001
        poly2[2]= {  8.33333333239095129568399345528033637663000e-0003,
                                 4.16666666394861917535640593963708222319000e-0002
        poly3[2]= { -1.98412623799797669279155177823009840396000e-0004,
                                -1.38888552656142867832756687736851681462000e-0003
        poly4[2]= {  2.75340362485427723764998762284833035111000e-0006,
                                 2.47851942368146079661812828945453052475900e-0005

/* Don't __ the following; acomp will handle it */
extern double fabs( double );
extern void __vlibm_vsincos_big( int, double *, int, double *, int, double *, in

/*
 * y[i*stridey] := sin( x[i*stridex] ), for i = 0..n.
 * c[i*stridec] := cos( x[i*stridex] ), for i = 0..n.
 *
 * Calls __vlibm_vsincos_big to handle all elts which have abs >~ 1.647e+06.
 * Argument reduction is done here for elts pi/4 < arg < 1.647e+06.
 *
 * elts < 2^-27 use the approximation 1.0 ~ cos(x).
 */
void
__vsincos( int n, double * restrict x, int stridex,
                                double * restrict y, int stridey,
                                double * restrict c, int stridec )
{
        double          x0_or_one[4], x1_or_one[4], x2_or_one[4];
        double          y0_or_zero[4], y1_or_zero[4], y2_or_zero[4];
        double          x0, x1, x2,
                        *py0, *py1, *py2,
                        *pc0, *pc1, *pc2,
                        *xsave, *ysave, *csave;
        unsigned        hx0, hx1, hx2, xsb0, xsb1, xsb2;
        int             i, biguns, nsave, sxsave, sysave, scsave;
        volatile int    v __GNU_UNUSED;
        nsave = n;
        xsave = x;
```

```
105         sxsave = stridex;
106         ysave = y;
107         sysave = stridey;
108         csave = c;
109         scsave = stridec;
110         biguns = 0;

112         do /* MAIN LOOP */
113         {

115                 /* Gotos here so _break_ exits MAIN LOOP. */
116 LOOP0:  /* Find first arg in right range. */
117                 xsb0 = HI(x); /* get most significant word */
118                 hx0 = xsb0 & ~0x80000000; /* mask off sign bit */
119                 if ( hx0 > 0x3fe921fb ) {
120                         /* Too big: arg reduction needed, so leave for second pa
121                         biguns = 1;
122                         x += stridex;
123                         y += stridey;
124                         c += stridec;
125                         i = 0;
126                         if ( --n <= 0 )
127                                 break;
128                         goto LOOP0;
129                 }
130                 if ( hx0 < 0x3e400000 ) {
131                         /* Too small.  cos x ~ 1, sin x ~ x. */
132                         v = *x;
133                         *c = 1.0;
134                         *y = *x;
135                         x += stridex;
136                         y += stridey;
137                         c += stridec;
138                         i = 0;
139                         if ( --n <= 0 )
140                                 break;
141                         goto LOOP0;
142                 }
143                 x0 = *x;
144                 py0 = y;
145                 pc0 = c;
146                 x += stridex;
147                 y += stridey;
148                 c += stridec;
149                 i = 1;
150                 if ( --n <= 0 )
151                         break;

153 LOOP1: /* Get second arg, same as above. */
154                 xsb1 = HI(x);
155                 hx1 = xsb1 & ~0x80000000;
156                 if ( hx1 > 0x3fe921fb )
157                 {
158                         biguns = 1;
159                         x += stridex;
160                         y += stridey;
161                         c += stridec;
162                         i = 1;
163                         if ( --n <= 0 )
164                                 break;
165                         goto LOOP1;
166                 }
167                 if ( hx1 < 0x3e400000 )
168                 {
169                         v = *x;
170                         *c = 1.0;
```



```
171                         *y = *x;
172                         x += stridex;
173                         y += stridey;
174                         c += stridec;
175                         i = 1;
176                         if ( --n <= 0 )
177                                 break;
178                         goto LOOP1;
179                 }
180                 x1 = *x;
181                 py1 = y;
182                 pc1 = c;
183                 x += stridex;
184                 y += stridey;
185                 c += stridec;
186                 i = 2;
187                 if ( --n <= 0 )
188                         break;

190 LOOP2: /* Get third arg, same as above. */
191                 xsb2 = HI(x);
192                 hx2 = xsb2 & ~0x80000000;
193                 if ( hx2 > 0x3fe921fb )
194                 {
195                         biguns = 1;
196                         x += stridex;
197                         y += stridey;
198                         c += stridec;
199                         i = 2;
200                         if ( --n <= 0 )
201                                 break;
202                         goto LOOP2;
203                 }
204                 if ( hx2 < 0x3e400000 )
205                 {
206                         v = *x;
207                         *c = 1.0;
208                         *y = *x;
209                         x += stridex;
210                         y += stridey;
211                         c += stridec;
212                         i = 2;
213                         if ( --n <= 0 )
214                                 break;
215                         goto LOOP2;
216                 }
217                 x2 = *x;
218                 py2 = y;
219                 pc2 = c;

221                 /*
222                  * 0x3fc40000 = 5/32 ~ 0.15625
223                  * Get msb after subtraction.  Will be 1 only if
224                  * hx0 - 5/32 is negative.
225                  */
226                 i = ( hx2 - 0x3fc40000 ) >> 31;
227                 i |= ( ( hx1 - 0x3fc40000 ) >> 30 ) & 2;
228                 i |= ( ( hx0 - 0x3fc40000 ) >> 29 ) & 4;
229                 switch ( i )
230                 {
231                         double          a1_0, a1_1, a1_2, a2_0, a2_1, a2_2;
232                         double          w0, w1, w2;
233                         double          t0, t1, t2, t1_0, t1_1, t1_2, t2_0, t2_1
234                         double          z0, z1, z2;
235                         unsigned        j0, j1, j2;
```

```
237                 case 0: /* All are > 5/32 */
238                         j0 = ( xsb0 + 0x4000 ) & 0xffff8000;
239                         j1 = ( xsb1 + 0x4000 ) & 0xffff8000;
240                         j2 = ( xsb2 + 0x4000 ) & 0xffff8000;

242                         HI(&t0) = j0;
243                         HI(&t1) = j1;
244                         HI(&t2) = j2;
245                         LO(&t0) = 0;
246                         LO(&t1) = 0;
247                         LO(&t2) = 0;

249                         x0 -= t0;
250                         x1 -= t1;
251                         x2 -= t2;

253                         z0 = x0 * x0;
254                         z1 = x1 * x1;
255                         z2 = x2 * x2;

257                         t0 = z0 * ( qq1 + z0 * qq2 );
258                         t1 = z1 * ( qq1 + z1 * qq2 );
259                         t2 = z2 * ( qq1 + z2 * qq2 );

261                         w0 = x0 * ( one + z0 * ( pp1 + z0 * pp2 ) );
262                         w1 = x1 * ( one + z1 * ( pp1 + z1 * pp2 ) );
263                         w2 = x2 * ( one + z2 * ( pp1 + z2 * pp2 ) );

265                         j0 = ( ( ( j0 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
266                         j1 = ( ( ( j1 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
267                         j2 = ( ( ( j2 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~

269                         xsb0 = ( xsb0 >> 30 ) & 2;
270                         xsb1 = ( xsb1 >> 30 ) & 2;
271                         xsb2 = ( xsb2 >> 30 ) & 2;

273                         a1_0 = __vlibm_TBL_sincos_hi[j0+xsb0]; /* sin_hi(t) */
274                         a1_1 = __vlibm_TBL_sincos_hi[j1+xsb1];
275                         a1_2 = __vlibm_TBL_sincos_hi[j2+xsb2];

277                         a2_0 = __vlibm_TBL_sincos_hi[j0+1];     /* cos_hi(t) */
278                         a2_1 = __vlibm_TBL_sincos_hi[j1+1];
279                         a2_2 = __vlibm_TBL_sincos_hi[j2+1];
280                                 /* cos_lo(t) */
281                         t2_0 = __vlibm_TBL_sincos_lo[j0+1] - ( a1_0*w0 - a2_0*t0
282                         t2_1 = __vlibm_TBL_sincos_lo[j1+1] - ( a1_1*w1 - a2_1*t1
283                         t2_2 = __vlibm_TBL_sincos_lo[j2+1] - ( a1_2*w2 - a2_2*t2

285                         *pc0 = a2_0 + t2_0;
286                         *pc1 = a2_1 + t2_1;
287                         *pc2 = a2_2 + t2_2;

289                         t1_0 = a2_0*w0 + a1_0*t0;
290                         t1_1 = a2_1*w1 + a1_1*t1;
291                         t1_2 = a2_2*w2 + a1_2*t2;

293                         t1_0 += __vlibm_TBL_sincos_lo[j0+xsb0]; /* sin_lo(t) */
294                         t1_1 += __vlibm_TBL_sincos_lo[j1+xsb1];
295                         t1_2 += __vlibm_TBL_sincos_lo[j2+xsb2];

297                         *py0 = a1_0 + t1_0;
298                         *py1 = a1_1 + t1_1;
299                         *py2 = a1_2 + t1_2;

301                         break;
```



```
303                 case 1:
304                         j0 = ( xsb0 + 0x4000 ) & 0xffff8000;
305                         j1 = ( xsb1 + 0x4000 ) & 0xffff8000;
306                         HI(&t0) = j0;
307                         HI(&t1) = j1;
308                         LO(&t0) = 0;
309                         LO(&t1) = 0;
310                         x0 -= t0;
311                         x1 -= t1;
312                         z0 = x0 * x0;
313                         z1 = x1 * x1;
314                         z2 = x2 * x2;
315                         t0 = z0 * ( qq1 + z0 * qq2 );
316                         t1 = z1 * ( qq1 + z1 * qq2 );
317                         t2 = z2 * ( poly3[1] + z2 * poly4[1] );
318                         w0 = x0 * ( one + z0 * ( pp1 + z0 * pp2 ) );
319                         w1 = x1 * ( one + z1 * ( pp1 + z1 * pp2 ) );
320                         t2 = z2 * ( poly1[1] + z2 * ( poly2[1] + t2 ) );
321                         j0 = ( ( ( j0 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
322                         j1 = ( ( ( j1 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
323                         xsb0 = ( xsb0 >> 30 ) & 2;
324                         xsb1 = ( xsb1 >> 30 ) & 2;

326                         a1_0 = __vlibm_TBL_sincos_hi[j0+xsb0]; /* sin_hi(t) */
327                         a1_1 = __vlibm_TBL_sincos_hi[j1+xsb1];

329                         a2_0 = __vlibm_TBL_sincos_hi[j0+1];     /* cos_hi(t) */
330                         a2_1 = __vlibm_TBL_sincos_hi[j1+1];
331                                 /* cos_lo(t) */
332                         t2_0 = __vlibm_TBL_sincos_lo[j0+1] - ( a1_0*w0 - a2_0*t0
333                         t2_1 = __vlibm_TBL_sincos_lo[j1+1] - ( a1_1*w1 - a2_1*t1

335                         *pc0 = a2_0 + t2_0;
336                         *pc1 = a2_1 + t2_1;
337                         *pc2 = one + t2;

339                         t1_0 = a2_0*w0 + a1_0*t0;
340                         t1_1 = a2_1*w1 + a1_1*t1;
341                         t2 = z2 * ( poly3[0] + z2 * poly4[0] );

343                         t1_0 += __vlibm_TBL_sincos_lo[j0+xsb0]; /* sin_lo(t) */
344                         t1_1 += __vlibm_TBL_sincos_lo[j1+xsb1];
345                         t2 = z2 * ( poly1[0] + z2 * ( poly2[0] + t2 ) );

347                         *py0 = a1_0 + t1_0;
348                         *py1 = a1_1 + t1_1;
349                         t2 = x2 + x2 * t2;
350                         *py2 = t2;

352                         break;

354                 case 2:
355                         j0 = ( xsb0 + 0x4000 ) & 0xffff8000;
356                         j2 = ( xsb2 + 0x4000 ) & 0xffff8000;
357                         HI(&t0) = j0;
358                         HI(&t2) = j2;
359                         LO(&t0) = 0;
360                         LO(&t2) = 0;
361                         x0 -= t0;
362                         x2 -= t2;
363                         z0 = x0 * x0;
364                         z1 = x1 * x1;
365                         z2 = x2 * x2;
366                         t0 = z0 * ( qq1 + z0 * qq2 );
367                         t1 = z1 * ( poly3[1] + z1 * poly4[1] );
368                         t2 = z2 * ( qq1 + z2 * qq2 );
```

```
369                     w0 = x0 * ( one + z0 * ( pp1 + z0 * pp2 ) );
370                     t1 = z1 * ( poly1[1] + z1 * ( poly2[1] + t1 ) );
371                     w2 = x2 * ( one + z2 * ( pp1 + z2 * pp2 ) );
372                     j0 = ( ( ( j0 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
373                     j2 = ( ( ( j2 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
374                     xsb0 = ( xsb0 >> 30 ) & 2;
375                     xsb2 = ( xsb2 >> 30 ) & 2;

377                     a1_0 = __vlibm_TBL_sincos_hi[j0+xsb0]; /* sin_hi(t) */
378                     a1_2 = __vlibm_TBL_sincos_hi[j2+xsb2];

380                     a2_0 = __vlibm_TBL_sincos_hi[j0+1];     /* cos_hi(t) */
381                     a2_2 = __vlibm_TBL_sincos_hi[j2+1];
382                              /* cos_lo(t) */
383                     t2_0 = __vlibm_TBL_sincos_lo[j0+1] - ( a1_0*w0 - a2_0*t0
384                     t2_2 = __vlibm_TBL_sincos_lo[j2+1] - ( a1_2*w2 - a2_2*t2

386                     *pc0 = a2_0 + t2_0;
387                     *pc1 = one + t1;
388                     *pc2 = a2_2 + t2_2;

390                     t1_0 = a2_0*w0 + a1_0*t0;
391                     t1 = z1 * ( poly3[0] + z1 * poly4[0] );
392                     t1_2 = a2_2*w2 + a1_2*t2;

394                     t1_0 += __vlibm_TBL_sincos_lo[j0+xsb0]; /* sin_lo(t) */
395                     t1 = z1 * ( poly1[0] + z1 * ( poly2[0] + t1 ) );
396                     t1_2 += __vlibm_TBL_sincos_lo[j2+xsb2];

398                     *py0 = a1_0 + t1_0;
399                     t1 = x1 + x1 * t1;
400                     *py1 = t1;
401                     *py2 = a1_2 + t1_2;

403                     break;

405             case 3:
406                     j0 = ( xsb0 + 0x4000 ) & 0xffff8000;
407                     HI(&t0) = j0;
408                     LO(&t0) = 0;
409                     x0 -= t0;
410                     z0 = x0 * x0;
411                     z1 = x1 * x1;
412                     z2 = x2 * x2;
413                     t0 = z0 * ( qq1 + z0 * qq2 );
414                     t1 = z1 * ( poly3[1] + z1 * poly4[1] );
415                     t2 = z2 * ( poly3[1] + z2 * poly4[1] );
416                     w0 = x0 * ( one + z0 * ( pp1 + z0 * pp2 ) );
417                     t1 = z1 * ( poly1[1] + z1 * ( poly2[1] + t1 ) );
418                     t2 = z2 * ( poly1[1] + z2 * ( poly2[1] + t2 ) );
419                     j0 = ( ( ( j0 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
420                     xsb0 = ( xsb0 >> 30 ) & 2;
421                     a1_0 = __vlibm_TBL_sincos_hi[j0+xsb0]; /* sin_hi(t) */

423                     a2_0 = __vlibm_TBL_sincos_hi[j0+1];     /* cos_hi(t) */

425                     t2_0 = __vlibm_TBL_sincos_lo[j0+1] - ( a1_0*w0 - a2_0*t0

427                     *pc0 = a2_0 + t2_0;
428                     *pc1 = one + t1;
429                     *pc2 = one + t2;

431                     t1_0 = a2_0*w0 + a1_0*t0;
432                     t1 = z1 * ( poly3[0] + z1 * poly4[0] );
433                     t2 = z2 * ( poly3[0] + z2 * poly4[0] );
```



```
435                     t1_0 += __vlibm_TBL_sincos_lo[j0+xsb0]; /* sin_lo(t) */
436                     t1 = z1 * ( poly1[0] + z1 * ( poly2[0] + t1 ) );
437                     t2 = z2 * ( poly1[0] + z2 * ( poly2[0] + t2 ) );

439                     *py0 = a1_0 + t1_0;
440                     t1 = x1 + x1 * t1;
441                     *py1 = t1;
442                     t2 = x2 + x2 * t2;
443                     *py2 = t2;

445                     break;

447             case 4:
448                     j1 = ( xsb1 + 0x4000 ) & 0xffff8000;
449                     j2 = ( xsb2 + 0x4000 ) & 0xffff8000;
450                     HI(&t1) = j1;
451                     HI(&t2) = j2;
452                     LO(&t1) = 0;
453                     LO(&t2) = 0;
454                     x1 -= t1;
455                     x2 -= t2;
456                     z0 = x0 * x0;
457                     z1 = x1 * x1;
458                     z2 = x2 * x2;
459                     t0 = z0 * ( poly3[1] + z0 * poly4[1] );
460                     t1 = z1 * ( qq1 + z1 * qq2 );
461                     t2 = z2 * ( qq1 + z2 * qq2 );
462                     t0 = z0 * ( poly1[1] + z0 * ( poly2[1] + t0 ) );
463                     w1 = x1 * ( one + z1 * ( pp1 + z1 * pp2 ) );
464                     w2 = x2 * ( one + z2 * ( pp1 + z2 * pp2 ) );
465                     j1 = ( ( ( j1 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
466                     j2 = ( ( ( j2 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
467                     xsb1 = ( xsb1 >> 30 ) & 2;
468                     xsb2 = ( xsb2 >> 30 ) & 2;

470                     a1_1 = __vlibm_TBL_sincos_hi[j1+xsb1];
471                     a1_2 = __vlibm_TBL_sincos_hi[j2+xsb2];

473                     a2_1 = __vlibm_TBL_sincos_hi[j1+1];
474                     a2_2 = __vlibm_TBL_sincos_hi[j2+1];
475                              /* cos_lo(t) */
476                     t2_1 = __vlibm_TBL_sincos_lo[j1+1] - ( a1_1*w1 - a2_1*t1
477                     t2_2 = __vlibm_TBL_sincos_lo[j2+1] - ( a1_2*w2 - a2_2*t2

479                     *pc0 = one + t0;
480                     *pc1 = a2_1 + t2_1;
481                     *pc2 = a2_2 + t2_2;

483                     t0 = z0 * ( poly3[0] + z0 * poly4[0] );
484                     t1_1 = a2_1*w1 + a1_1*t1;
485                     t1_2 = a2_2*w2 + a1_2*t2;

487                     t0 = z0 * ( poly1[0] + z0 * ( poly2[0] + t0 ) );
488                     t1_1 += __vlibm_TBL_sincos_lo[j1+xsb1];
489                     t1_2 += __vlibm_TBL_sincos_lo[j2+xsb2];

491                     t0 = x0 + x0 * t0;
492                     *py0 = t0;
493                     *py1 = a1_1 + t1_1;
494                     *py2 = a1_2 + t1_2;

496                     break;

498             case 5:
499                     j1 = ( xsb1 + 0x4000 ) & 0xffff8000;
500                     HI(&t1) = j1;
```

```
501                         LO(&t1) = 0;
502                         x1 -= t1;
503                         z0 = x0 * x0;
504                         z1 = x1 * x1;
505                         z2 = x2 * x2;
506                         t0 = z0 * ( poly3[1] + z0 * poly4[1] );
507                         t1 = z1 * ( qq1 + z1 * qq2 );
508                         t2 = z2 * ( poly3[1] + z2 * poly4[1] );
509                         t0 = z0 * ( poly1[1] + z0 * ( poly2[1] + t0 ) );
510                         w1 = x1 * ( one + z1 * ( pp1 + z1 * pp2 ) );
511                         t2 = z2 * ( poly1[1] + z2 * ( poly2[1] + t2 ) );
512                         j1 = ( ( ( j1 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
513                         xsb1 = ( xsb1 >> 30 ) & 2;

515                         a1_1 = __vlibm_TBL_sincos_hi[j1+xsb1];

517                         a2_1 = __vlibm_TBL_sincos_hi[j1+1];

519                         t2_1 = __vlibm_TBL_sincos_lo[j1+1] - ( a1_1*w1 - a2_1*t1

521                         *pc0 = one + t0;
522                         *pc1 = a2_1 + t2_1;
523                         *pc2 = one + t2;

525                         t0 = z0 * ( poly3[0] + z0 * poly4[0] );
526                         t1_1 = a2_1*w1 + a1_1*t1;
527                         t2 = z2 * ( poly3[0] + z2 * poly4[0] );

529                         t0 = z0 * ( poly1[0] + z0 * ( poly2[0] + t0 ) );
530                         t1_1 += __vlibm_TBL_sincos_lo[j1+xsb1];
531                         t2 = z2 * ( poly1[0] + z2 * ( poly2[0] + t2 ) );

533                         t0 = x0 + x0 * t0;
534                         *py0 = t0;
535                         *py1 = a1_1 + t1_1;
536                         t2 = x2 + x2 * t2;
537                         *py2 = t2;

539                         break;

541                 case 6:
542                         j2 = ( xsb2 + 0x4000 ) & 0xffff8000;
543                         HI(&t2) = j2;
544                         LO(&t2) = 0;
545                         x2 -= t2;
546                         z0 = x0 * x0;
547                         z1 = x1 * x1;
548                         z2 = x2 * x2;
549                         t0 = z0 * ( poly3[1] + z0 * poly4[1] );
550                         t1 = z1 * ( poly3[1] + z1 * poly4[1] );
551                         t2 = z2 * ( qq1 + z2 * qq2 );
552                         t0 = z0 * ( poly1[1] + z0 * ( poly2[1] + t0 ) );
553                         t1 = z1 * ( poly1[1] + z1 * ( poly2[1] + t1 ) );
554                         w2 = x2 * ( one + z2 * ( pp1 + z2 * pp2 ) );
555                         j2 = ( ( ( j2 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
556                         xsb2 = ( xsb2 >> 30 ) & 2;
557                         a1_2 = __vlibm_TBL_sincos_hi[j2+xsb2];

559                         a2_2 = __vlibm_TBL_sincos_hi[j2+1];

561                         t2_2 = __vlibm_TBL_sincos_lo[j2+1] - ( a1_2*w2 - a2_2*t2

563                         *pc0 = one + t0;
564                         *pc1 = one + t1;
565                         *pc2 = a2_2 + t2_2;
```



```
567                         t0 = z0 * ( poly3[0] + z0 * poly4[0] );
568                         t1 = z1 * ( poly3[0] + z1 * poly4[0] );
569                         t1_2 = a2_2*w2 + a1_2*t2;

571                         t0 = z0 * ( poly1[0] + z0 * ( poly2[0] + t0 ) );
572                         t1 = z1 * ( poly1[0] + z1 * ( poly2[0] + t1 ) );
573                         t1_2 += __vlibm_TBL_sincos_lo[j2+xsb2];

575                         t0 = x0 + x0 * t0;
576                         *py0 = t0;
577                         t1 = x1 + x1 * t1;
578                         *py1 = t1;
579                         *py2 = a1_2 + t1_2;

581                         break;

583                 case 7: /* All are < 5/32 */
584                         z0 = x0 * x0;
585                         z1 = x1 * x1;
586                         z2 = x2 * x2;
587                         t0 = z0 * ( poly3[1] + z0 * poly4[1] );
588                         t1 = z1 * ( poly3[1] + z1 * poly4[1] );
589                         t2 = z2 * ( poly3[1] + z2 * poly4[1] );
590                         t0 = z0 * ( poly1[1] + z0 * ( poly2[1] + t0 ) );
591                         t1 = z1 * ( poly1[1] + z1 * ( poly2[1] + t1 ) );
592                         t2 = z2 * ( poly1[1] + z2 * ( poly2[1] + t2 ) );
593                         *pc0 = one + t0;
594                         *pc1 = one + t1;
595                         *pc2 = one + t2;
596                         t0 = z0 * ( poly3[0] + z0 * poly4[0] );
597                         t1 = z1 * ( poly3[0] + z1 * poly4[0] );
598                         t2 = z2 * ( poly3[0] + z2 * poly4[0] );
599                         t0 = z0 * ( poly1[0] + z0 * ( poly2[0] + t0 ) );
600                         t1 = z1 * ( poly1[0] + z1 * ( poly2[0] + t1 ) );
601                         t2 = z2 * ( poly1[0] + z2 * ( poly2[0] + t2 ) );
602                         t0 = x0 + x0 * t0;
603                         t1 = x1 + x1 * t1;
604                         t2 = x2 + x2 * t2;
605                         *py0 = t0;
606                         *py1 = t1;
607                         *py2 = t2;
608                         break;
609                 }

611                 x += stridex;
612                 y += stridey;
613                 c += stridec;
614                 i = 0;
615         } while ( --n > 0 ); /* END MAIN LOOP */

617         /*
618          * CLEAN UP last 0, 1, or 2 elts.
619          */
620         if ( i > 0 ) /* Clean up elts at tail.  i < 3. */
621         {
622                 double          a1_0, a1_1, a2_0, a2_1;
623                 double          w0, w1;
624                 double          t0, t1, t1_0, t1_1, t2_0, t2_1;
625                 double          z0, z1;
626                 unsigned        j0, j1;

628                 if ( i > 1 )
629                 {
630                         if ( hx1 < 0x3fc40000 )
631                         {
632                                 z1 = x1 * x1;
```

```
633                             t1 = z1 * ( poly3[1] + z1 * poly4[1] );
634                             t1 = z1 * ( poly1[1] + z1 * ( poly2[1] + t1 ) );
635                             t1 = one + t1;
636                             *pc1 = t1;
637                             t1 = z1 * ( poly3[0] + z1 * poly4[0] );
638                             t1 = z1 * ( poly1[0] + z1 * ( poly2[0] + t1 ) );
639                             t1 = x1 + x1 * t1;
640                             *py1 = t1;
641                     }
642                     else
643                     {
644                             j1 = ( xsb1 + 0x4000 ) & 0xffff8000;
645                             HI(&t1) = j1;
646                             LO(&t1) = 0;
647                             x1 -= t1;
648                             z1 = x1 * x1;
649                             t1 = z1 * ( qq1 + z1 * qq2 );
650                             w1 = x1 * ( one + z1 * ( pp1 + z1 * pp2 ) );
651                             j1 = ( ( ( j1 & ~0x80000000 ) - 0x3fc40000 ) >>
652                             xsb1 = ( xsb1 >> 30 ) & 2;
653                             a1_1 = __vlibm_TBL_sincos_hi[j1+xsb1];
654                             a2_1 = __vlibm_TBL_sincos_hi[j1+1];
655                             t2_1 = __vlibm_TBL_sincos_lo[j1+1] - ( a1_1*w1 -
656                             *pc1 = a2_1 + t2_1;
657                             t1_1 = a2_1*w1 + a1_1*t1;
658                             t1_1 += __vlibm_TBL_sincos_lo[j1+xsb1];
659                             *py1 = a1_1 + t1_1;
660                     }
661             }
662             if ( hx0 < 0x3fc40000 )
663             {
664                     z0 = x0 * x0;
665                     t0 = z0 * ( poly3[1] + z0 * poly4[1] );
666                     t0 = z0 * ( poly1[1] + z0 * ( poly2[1] + t0 ) );
667                     t0 = one + t0;
668                     *pc0 = t0;
669                     t0 = z0 * ( poly3[0] + z0 * poly4[0] );
670                     t0 = z0 * ( poly1[0] + z0 * ( poly2[0] + t0 ) );
671                     t0 = x0 + x0 * t0;
672                     *py0 = t0;
673             }
674             else
675             {
676                     j0 = ( xsb0 + 0x4000 ) & 0xffff8000;
677                     HI(&t0) = j0;
678                     LO(&t0) = 0;
679                     x0 -= t0;
680                     z0 = x0 * x0;
681                     t0 = z0 * ( qq1 + z0 * qq2 );
682                     w0 = x0 * ( one + z0 * ( pp1 + z0 * pp2 ) );
683                     j0 = ( ( ( j0 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
684                     xsb0 = ( xsb0 >> 30 ) & 2;
685                     a1_0 = __vlibm_TBL_sincos_hi[j0+xsb0]; /* sin_hi(t) */
686                     a2_0 = __vlibm_TBL_sincos_hi[j0+1];     /* cos_hi(t) */
687                     t2_0 = __vlibm_TBL_sincos_lo[j0+1] - ( a1_0*w0 - a2_0*t0
688                     *pc0 = a2_0 + t2_0;
689                     t1_0 = a2_0*w0 + a1_0*t0;
690                     t1_0 += __vlibm_TBL_sincos_lo[j0+xsb0]; /* sin_lo(t) */
691                     *py0 = a1_0 + t1_0;
692             }
693         } /* END CLEAN UP */

695         if ( !biguns )
696                 return;

698         /*
```



```
699          * Take care of BIGUNS.
700          */
701         n = nsave;
702         x = xsave;
703         stridex = sxsave;
704         y = ysave;
705         stridey = sysave;
706         c = csave;
707         stridec = scsave;
708         biguns = 0;

710         x0_or_one[1] = 1.0;
711         x1_or_one[1] = 1.0;
712         x2_or_one[1] = 1.0;
713         x0_or_one[3] = -1.0;
714         x1_or_one[3] = -1.0;
715         x2_or_one[3] = -1.0;
716         y0_or_zero[1] = 0.0;
717         y1_or_zero[1] = 0.0;
718         y2_or_zero[1] = 0.0;
719         y0_or_zero[3] = 0.0;
720         y1_or_zero[3] = 0.0;
721         y2_or_zero[3] = 0.0;

723         do
724         {
725                 double          fn0, fn1, fn2, a0, a1, a2, w0, w1, w2, y0, y1, y
726                 unsigned        hx;
727                 int                     n0, n1, n2;

729                 /*
730                  * Find 3 more to work on: Not already done, not too big.
731                  */
732 loop0:
733                 hx = HI(x);
734                 xsb0 = hx >> 31;
735                 hx &= ~0x80000000;
736                 if ( hx <= 0x3fe921fb ) /* Done above. */
737                 {
738                         x += stridex;
739                         y += stridey;
740                         c += stridec;
741                         i = 0;
742                         if ( --n <= 0 )
743                                 break;
744                         goto loop0;
745                 }
746                 if ( hx > 0x413921fb ) /* (1.6471e+06) Too big: leave it. */
747                 {
748                         if ( hx >= 0x7ff00000 ) /* Inf or NaN */
749                         {
750                                 x0 = *x;
751                                 *y = x0 - x0;
752                                 *c = x0 - x0;
753                         }
754                         else {
755                                 biguns = 1;
756                         }
757                         x += stridex;
758                         y += stridey;
759                         c += stridec;
760                         i = 0;
761                         if ( --n <= 0 )
762                                 break;
763                         goto loop0;
764                 }
```

```
765                 x0 = *x;
766                 py0 = y;
767                 pc0 = c;
768                 x += stridex;
769                 y += stridey;
770                 c += stridec;
771                 i = 1;
772                 if ( --n <= 0 )
773                         break;

775 loop1:
776                 hx = HI(x);
777                 xsb1 = hx >> 31;
778                 hx &= ~0x80000000;
779                 if ( hx <= 0x3fe921fb )
780                 {
781                         x += stridex;
782                         y += stridey;
783                         c += stridec;
784                         i = 1;
785                         if ( --n <= 0 )
786                                 break;
787                         goto loop1;
788                 }
789                 if ( hx > 0x413921fb )
790                 {
791                         if ( hx >= 0x7ff00000 )
792                         {
793                                 x1 = *x;
794                                 *y = x1 - x1;
795                                 *c = x1 - x1;
796                         }
797                         else {
798                                 biguns = 1;
799                         }
800                         x += stridex;
801                         y += stridey;
802                         c += stridec;
803                         i = 1;
804                         if ( --n <= 0 )
805                                 break;
806                         goto loop1;
807                 }
808                 x1 = *x;
809                 py1 = y;
810                 pc1 = c;
811                 x += stridex;
812                 y += stridey;
813                 c += stridec;
814                 i = 2;
815                 if ( --n <= 0 )
816                         break;

818 loop2:
819                 hx = HI(x);
820                 xsb2 = hx >> 31;
821                 hx &= ~0x80000000;
822                 if ( hx <= 0x3fe921fb )
823                 {
824                         x += stridex;
825                         y += stridey;
826                         c += stridec;
827                         i = 2;
828                         if ( --n <= 0 )
829                                 break;
830                         goto loop2;
```



```
831                 }
832                 if ( hx > 0x413921fb )
833                 {
834                         if ( hx >= 0x7ff00000 )
835                         {
836                                 x2 = *x;
837                                 *y = x2 - x2;
838                                 *c = x2 - x2;
839                         }
840                         else {
841                                 biguns = 1;
842                         }
843                         x += stridex;
844                         y += stridey;
845                         c += stridec;
846                         i = 2;
847                         if ( --n <= 0 )
848                                 break;
849                         goto loop2;
850                 }
851                 x2 = *x;
852                 py2 = y;
853                 pc2 = c;

855                 n0 = (int) ( x0 * invpio2 + half[xsb0] );
856                 n1 = (int) ( x1 * invpio2 + half[xsb1] );
857                 n2 = (int) ( x2 * invpio2 + half[xsb2] );
858                 fn0 = (double) n0;
859                 fn1 = (double) n1;
860                 fn2 = (double) n2;
861                 n0 &= 3;
862                 n1 &= 3;
863                 n2 &= 3;
864                 a0 = x0 - fn0 * pio2_1;
865                 a1 = x1 - fn1 * pio2_1;
866                 a2 = x2 - fn2 * pio2_1;
867                 w0 = fn0 * pio2_2;
868                 w1 = fn1 * pio2_2;
869                 w2 = fn2 * pio2_2;
870                 x0 = a0 - w0;
871                 x1 = a1 - w1;
872                 x2 = a2 - w2;
873                 y0 = ( a0 - x0 ) - w0;
874                 y1 = ( a1 - x1 ) - w1;
875                 y2 = ( a2 - x2 ) - w2;
876                 a0 = x0;
877                 a1 = x1;
878                 a2 = x2;
879                 w0 = fn0 * pio2_3 - y0;
880                 w1 = fn1 * pio2_3 - y1;
881                 w2 = fn2 * pio2_3 - y2;
882                 x0 = a0 - w0;
883                 x1 = a1 - w1;
884                 x2 = a2 - w2;
885                 y0 = ( a0 - x0 ) - w0;
886                 y1 = ( a1 - x1 ) - w1;
887                 y2 = ( a2 - x2 ) - w2;
888                 a0 = x0;
889                 a1 = x1;
890                 a2 = x2;
891                 w0 = fn0 * pio2_3t - y0;
892                 w1 = fn1 * pio2_3t - y1;
893                 w2 = fn2 * pio2_3t - y2;
894                 x0 = a0 - w0;
895                 x1 = a1 - w1;
896                 x2 = a2 - w2;
```

```
897                 y0 = ( a0 - x0 ) - w0;
898                 y1 = ( a1 - x1 ) - w1;
899                 y2 = ( a2 - x2 ) - w2;
900                 xsb2 = HI(&x2);
901                 i = ( ( xsb2 & ~0x80000000 ) - 0x3fc40000 ) >> 31;
902                 xsb1 = HI(&x1);
903                 i |= ( ( ( xsb1 & ~0x80000000 ) - 0x3fc40000 ) >> 30 ) & 2;
904                 xsb0 = HI(&x0);
905                 i |= ( ( ( xsb0 & ~0x80000000 ) - 0x3fc40000 ) >> 29 ) & 4;
906                 switch ( i )
907                 {
908                         double          a1_0, a1_1, a1_2, a2_0, a2_1, a2_2;
909                         double          t0, t1, t2, t1_0, t1_1, t1_2, t2_0, t2_1
910                         double          z0, z1, z2;
911                         unsigned        j0, j1, j2;
912
913                 case 0:
914                         j0 = ( xsb0 + 0x4000 ) & 0xffff8000;
915                         j1 = ( xsb1 + 0x4000 ) & 0xffff8000;
916                         j2 = ( xsb2 + 0x4000 ) & 0xffff8000;
917                         HI(&t0) = j0;
918                         HI(&t1) = j1;
919                         HI(&t2) = j2;
920                         LO(&t0) = 0;
921                         LO(&t1) = 0;
922                         LO(&t2) = 0;
923                         x0 = ( x0 - t0 ) + y0;
924                         x1 = ( x1 - t1 ) + y1;
925                         x2 = ( x2 - t2 ) + y2;
926                         z0 = x0 * x0;
927                         z1 = x1 * x1;
928                         z2 = x2 * x2;
929                         t0 = z0 * ( qq1 + z0 * qq2 );
930                         t1 = z1 * ( qq1 + z1 * qq2 );
931                         t2 = z2 * ( qq1 + z2 * qq2 );
932                         w0 = x0 * ( one + z0 * ( pp1 + z0 * pp2 ) );
933                         w1 = x1 * ( one + z1 * ( pp1 + z1 * pp2 ) );
934                         w2 = x2 * ( one + z2 * ( pp1 + z2 * pp2 ) );
935                         j0 = ( ( ( j0 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
936                         j1 = ( ( ( j1 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
937                         j2 = ( ( ( j2 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
938                         xsb0 = ( xsb0 >> 30 ) & 2;
939                         xsb1 = ( xsb1 >> 30 ) & 2;
940                         xsb2 = ( xsb2 >> 30 ) & 2;
941                         n0 ^= ( xsb0 & ~( n0 << 1 ) );
942                         n1 ^= ( xsb1 & ~( n1 << 1 ) );
943                         n2 ^= ( xsb2 & ~( n2 << 1 ) );
944                         xsb0 |= 1;
945                         xsb1 |= 1;
946                         xsb2 |= 1;
947
948                         a1_0 = __vlibm_TBL_sincos_hi[j0+n0];
949                         a1_1 = __vlibm_TBL_sincos_hi[j1+n1];
950                         a1_2 = __vlibm_TBL_sincos_hi[j2+n2];
951
952                         a2_0 = __vlibm_TBL_sincos_hi[j0+((n0+xsb0)&3)];
953                         a2_1 = __vlibm_TBL_sincos_hi[j1+((n1+xsb1)&3)];
954                         a2_2 = __vlibm_TBL_sincos_hi[j2+((n2+xsb2)&3)];
955
956                         t2_0 = __vlibm_TBL_sincos_lo[j0+((n0+xsb0)&3)] - ( a1_0*
957                         t2_1 = __vlibm_TBL_sincos_lo[j1+((n1+xsb1)&3)] - ( a1_1*
958                         t2_2 = __vlibm_TBL_sincos_lo[j2+((n2+xsb2)&3)] - ( a1_2*
959
960                         w0 *= a2_0;
961                         w1 *= a2_1;
962                         w2 *= a2_2;
```



```
964                         *pc0 = a2_0 + t2_0;
965                         *pc1 = a2_1 + t2_1;
966                         *pc2 = a2_2 + t2_2;
967
968                         t1_0 = w0 + a1_0*t0;
969                         t1_1 = w1 + a1_1*t1;
970                         t1_2 = w2 + a1_2*t2;
971
972                         t1_0 += __vlibm_TBL_sincos_lo[j0+n0];
973                         t1_1 += __vlibm_TBL_sincos_lo[j1+n1];
974                         t1_2 += __vlibm_TBL_sincos_lo[j2+n2];
975
976                         *py0 = a1_0 + t1_0;
977                         *py1 = a1_1 + t1_1;
978                         *py2 = a1_2 + t1_2;
979
980                         break;
981
982                 case 1:
983                         j0 = ( xsb0 + 0x4000 ) & 0xffff8000;
984                         j1 = ( xsb1 + 0x4000 ) & 0xffff8000;
985                         j2 = n2 & 1;
986                         HI(&t0) = j0;
987                         HI(&t1) = j1;
988                         LO(&t0) = 0;
989                         LO(&t1) = 0;
990                         x2_or_one[0] = x2;
991                         x2_or_one[2] = -x2;
992                         x0 = ( x0 - t0 ) + y0;
993                         x1 = ( x1 - t1 ) + y1;
994                         y2_or_zero[0] = y2;
995                         y2_or_zero[2] = -y2;
996                         z0 = x0 * x0;
997                         z1 = x1 * x1;
998                         z2 = x2 * x2;
999                         t0 = z0 * ( qq1 + z0 * qq2 );
1000                        t1 = z1 * ( qq1 + z1 * qq2 );
1001                        t2 = z2 * ( poly3[j2] + z2 * poly4[j2] );
1002                        w0 = x0 * ( one + z0 * ( pp1 + z0 * pp2 ) );
1003                        w1 = x1 * ( one + z1 * ( pp1 + z1 * pp2 ) );
1004                        t2 = z2 * ( poly1[j2] + z2 * ( poly2[j2] + t2 ) );
1005                        j0 = ( ( ( j0 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
1006                        j1 = ( ( ( j1 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
1007                        xsb0 = ( xsb0 >> 30 ) & 2;
1008                        xsb1 = ( xsb1 >> 30 ) & 2;
1009                        n0 ^= ( xsb0 & ~( n0 << 1 ) );
1010                        n1 ^= ( xsb1 & ~( n1 << 1 ) );
1011                        xsb0 |= 1;
1012                        xsb1 |= 1;
1013                        a1_0 = __vlibm_TBL_sincos_hi[j0+n0];
1014                        a1_1 = __vlibm_TBL_sincos_hi[j1+n1];
1015
1016                        a2_0 = __vlibm_TBL_sincos_hi[j0+((n0+xsb0)&3)];
1017                        a2_1 = __vlibm_TBL_sincos_hi[j1+((n1+xsb1)&3)];
1018
1019                        t2_0 = __vlibm_TBL_sincos_lo[j0+((n0+xsb0)&3)] - ( a1_0*
1020                        t2_1 = __vlibm_TBL_sincos_lo[j1+((n1+xsb1)&3)] - ( a1_1*
1021                        t2 = x2_or_one[n2] + ( y2_or_zero[n2] + x2_or_one[n2] *
1022
1023                        *pc0 = a2_0 + t2_0;
1024                        *pc1 = a2_1 + t2_1;
1025                        *py2 = t2;
1026
1027                        n2 = (n2 + 1) & 3;
1028                        j2 = (j2 + 1) & 1;
```

```
1029                        t2 = z2 * ( poly3[j2] + z2 * poly4[j2] );
1030
1031                        t1_0 = a2_0*w0 + a1_0*t0;
1032                        t1_1 = a2_1*w1 + a1_1*t1;
1033                        t2 = z2 * ( poly1[j2] + z2 * ( poly2[j2] + t2 ) );
1034
1035                        t1_0 += __vlibm_TBL_sincos_lo[j0+n0];
1036                        t1_1 += __vlibm_TBL_sincos_lo[j1+n1];
1037                        t2 = x2_or_one[n2] + ( y2_or_zero[n2] + x2_or_one[n2] *
1038
1039                        *py0 = a1_0 + t1_0;
1040                        *py1 = a1_1 + t1_1;
1041                        *pc2 = t2;
1042
1043                        break;
1044
1045                case 2:
1046                        j0 = ( xsb0 + 0x4000 ) & 0xffff8000;
1047                        j1 = n1 & 1;
1048                        j2 = ( xsb2 + 0x4000 ) & 0xffff8000;
1049                        HI(&t0) = j0;
1050                        HI(&t2) = j2;
1051                        LO(&t0) = 0;
1052                        LO(&t2) = 0;
1053                        x1_or_one[0] = x1;
1054                        x1_or_one[2] = -x1;
1055                        x0 = ( x0 - t0 ) + y0;
1056                        y1_or_zero[0] = y1;
1057                        y1_or_zero[2] = -y1;
1058                        x2 = ( x2 - t2 ) + y2;
1059                        z0 = x0 * x0;
1060                        z1 = x1 * x1;
1061                        z2 = x2 * x2;
1062                        t0 = z0 * ( qq1 + z0 * qq2 );
1063                        t1 = z1 * ( poly3[j1] + z1 * poly4[j1] );
1064                        t2 = z2 * ( qq1 + z2 * qq2 );
1065                        w0 = x0 * ( one + z0 * ( pp1 + z0 * pp2 ) );
1066                        t1 = z1 * ( poly1[j1] + z1 * ( poly2[j1] + t1 ) );
1067                        w2 = x2 * ( one + z2 * ( pp1 + z2 * pp2 ) );
1068                        j0 = ( ( ( j0 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
1069                        j2 = ( ( ( j2 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
1070                        xsb0 = ( xsb0 >> 30 ) & 2;
1071                        xsb2 = ( xsb2 >> 30 ) & 2;
1072                        n0 ^= ( xsb0 & ~( n0 << 1 ) );
1073                        n2 ^= ( xsb2 & ~( n2 << 1 ) );
1074                        xsb0 |= 1;
1075                        xsb2 |= 1;
1076
1077                        a1_0 = __vlibm_TBL_sincos_hi[j0+n0];
1078                        a1_2 = __vlibm_TBL_sincos_hi[j2+n2];
1079
1080                        a2_0 = __vlibm_TBL_sincos_hi[j0+((n0+xsb0)&3)];
1081                        a2_2 = __vlibm_TBL_sincos_hi[j2+((n2+xsb2)&3)];
1082
1083                        t2_0 = __vlibm_TBL_sincos_lo[j0+((n0+xsb0)&3)] - ( a1_0*
1084                        t1 = x1_or_one[n1] + ( y1_or_zero[n1] + x1_or_one[n1] *
1085                        t2_2 = __vlibm_TBL_sincos_lo[j2+((n2+xsb2)&3)] - ( a1_2*
1086
1087                        *pc0 = a2_0 + t2_0;
1088                        *py1 = t1;
1089                        *pc2 = a2_2 + t2_2;
1090
1091                        n1 = (n1 + 1) & 3;
1092                        j1 = (j1 + 1) & 1;
1093                        t1 = z1 * ( poly3[j1] + z1 * poly4[j1] );
```



```
1095                        t1_0 = a2_0*w0 + a1_0*t0;
1096                        t1 = z1 * ( poly1[j1] + z1 * ( poly2[j1] + t1 ) );
1097                        t1_2 = a2_2*w2 + a1_2*t2;
1098
1099                        t1_0 += __vlibm_TBL_sincos_lo[j0+n0];
1100                        t1 = x1_or_one[n1] + ( y1_or_zero[n1] + x1_or_one[n1] *
1101                        t1_2 += __vlibm_TBL_sincos_lo[j2+n2];
1102
1103                        *py0 = a1_0 + t1_0;
1104                        *pc1 = t1;
1105                        *py2 = a1_2 + t1_2;
1106
1107                        break;
1108
1109                case 3:
1110                        j0 = ( xsb0 + 0x4000 ) & 0xffff8000;
1111                        j1 = n1 & 1;
1112                        j2 = n2 & 1;
1113                        HI(&t0) = j0;
1114                        LO(&t0) = 0;
1115                        x1_or_one[0] = x1;
1116                        x1_or_one[2] = -x1;
1117                        x2_or_one[0] = x2;
1118                        x2_or_one[2] = -x2;
1119                        x0 = ( x0 - t0 ) + y0;
1120                        y1_or_zero[0] = y1;
1121                        y1_or_zero[2] = -y1;
1122                        y2_or_zero[0] = y2;
1123                        y2_or_zero[2] = -y2;
1124                        z0 = x0 * x0;
1125                        z1 = x1 * x1;
1126                        z2 = x2 * x2;
1127                        t0 = z0 * ( qq1 + z0 * qq2 );
1128                        t1 = z1 * ( poly3[j1] + z1 * poly4[j1] );
1129                        t2 = z2 * ( poly3[j2] + z2 * poly4[j2] );
1130                        w0 = x0 * ( one + z0 * ( pp1 + z0 * pp2 ) );
1131                        t1 = z1 * ( poly1[j1] + z1 * ( poly2[j1] + t1 ) );
1132                        t2 = z2 * ( poly1[j2] + z2 * ( poly2[j2] + t2 ) );
1133                        j0 = ( ( ( j0 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
1134                        xsb0 = ( xsb0 >> 30 ) & 2;
1135                        n0 ^= ( xsb0 & ~( n0 << 1 ) );
1136                        xsb0 |= 1;
1137
1138                        a1_0 = __vlibm_TBL_sincos_hi[j0+n0];
1139                        a2_0 = __vlibm_TBL_sincos_hi[j0+((n0+xsb0)&3)];
1140
1141                        t2_0 = __vlibm_TBL_sincos_lo[j0+((n0+xsb0)&3)] - ( a1_0*
1142                        t1 = x1_or_one[n1] + ( y1_or_zero[n1] + x1_or_one[n1] *
1143                        t2 = x2_or_one[n2] + ( y2_or_zero[n2] + x2_or_one[n2] *
1144
1145                        *pc0 = a2_0 + t2_0;
1146                        *py1 = t1;
1147                        *py2 = t2;
1148
1149                        n1 = (n1 + 1) & 3;
1150                        n2 = (n2 + 1) & 3;
1151                        j1 = (j1 + 1) & 1;
1152                        j2 = (j2 + 1) & 1;
1153
1154                        t1_0 = a2_0*w0 + a1_0*t0;
1155                        t1 = z1 * ( poly3[j1] + z1 * poly4[j1] );
1156                        t2 = z2 * ( poly3[j2] + z2 * poly4[j2] );
1157
1158                        t1_0 += __vlibm_TBL_sincos_lo[j0+n0];
1159                        t1 = z1 * ( poly1[j1] + z1 * ( poly2[j1] + t1 ) );
1160                        t2 = z2 * ( poly1[j2] + z2 * ( poly2[j2] + t2 ) );
```

```
1162                        t1 = x1_or_one[n1] + ( y1_or_zero[n1] + x1_or_one[n1] *
1163                        t2 = x2_or_one[n2] + ( y2_or_zero[n2] + x2_or_one[n2] *

1165                        *py0 = a1_0 + t1_0;
1166                        *pc1 = t1;
1167                        *pc2 = t2;

1169                        break;

1171                case 4:
1172                        j0 = n0 & 1;
1173                        j1 = ( xsb1 + 0x4000 ) & 0xffff8000;
1174                        j2 = ( xsb2 + 0x4000 ) & 0xffff8000;
1175                        HI(&t1) = j1;
1176                        HI(&t2) = j2;
1177                        LO(&t1) = 0;
1178                        LO(&t2) = 0;
1179                        x0_or_one[0] = x0;
1180                        x0_or_one[2] = -x0;
1181                        y0_or_zero[0] = y0;
1182                        y0_or_zero[2] = -y0;
1183                        x1 = ( x1 - t1 ) + y1;
1184                        x2 = ( x2 - t2 ) + y2;
1185                        z0 = x0 * x0;
1186                        z1 = x1 * x1;
1187                        z2 = x2 * x2;
1188                        t0 = z0 * ( poly3[j0] + z0 * poly4[j0] );
1189                        t1 = z1 * ( qq1 + z1 * qq2 );
1190                        t2 = z2 * ( qq1 + z2 * qq2 );
1191                        t0 = z0 * ( poly1[j0] + z0 * ( poly2[j0] + t0 ) );
1192                        w1 = x1 * ( one + z1 * ( pp1 + z1 * pp2 ) );
1193                        w2 = x2 * ( one + z2 * ( pp1 + z2 * pp2 ) );
1194                        j1 = ( ( ( j1 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
1195                        j2 = ( ( ( j2 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
1196                        xsb1 = ( xsb1 >> 30 ) & 2;
1197                        xsb2 = ( xsb2 >> 30 ) & 2;
1198                        n1 ^= ( xsb1 & ~( n1 << 1 ) );
1199                        n2 ^= ( xsb2 & ~( n2 << 1 ) );
1200                        xsb1 |= 1;
1201                        xsb2 |= 1;

1203                        a1_1 = __vlibm_TBL_sincos_hi[j1+n1];
1204                        a1_2 = __vlibm_TBL_sincos_hi[j2+n2];

1206                        a2_1 = __vlibm_TBL_sincos_hi[j1+((n1+xsb1)&3)];
1207                        a2_2 = __vlibm_TBL_sincos_hi[j2+((n2+xsb2)&3)];

1209                        t0 = x0_or_one[n0] + ( y0_or_zero[n0] + x0_or_one[n0] *
1210                        t2_1 = __vlibm_TBL_sincos_lo[j1+((n1+xsb1)&3)] - ( a1_1*
1211                        t2_2 = __vlibm_TBL_sincos_lo[j2+((n2+xsb2)&3)] - ( a1_2*

1213                        *py0 = t0;
1214                        *pc1 = a2_1 + t2_1;
1215                        *pc2 = a2_2 + t2_2;

1217                        n0 = (n0 + 1) & 3;
1218                        j0 = (j0 + 1) & 1;
1219                        t0 = z0 * ( poly3[j0] + z0 * poly4[j0] );

1221                        t0 = z0 * ( poly1[j0] + z0 * ( poly2[j0] + t0 ) );
1222                        t1_1 = a2_1*w1 + a1_1*t1;
1223                        t1_2 = a2_2*w2 + a1_2*t2;

1225                        t0 = x0_or_one[n0] + ( y0_or_zero[n0] + x0_or_one[n0] *
1226                        t1_1 += __vlibm_TBL_sincos_lo[j1+n1];
```



```
1227                        t1_2 += __vlibm_TBL_sincos_lo[j2+n2];

1229                        *py1 = a1_1 + t1_1;
1230                        *py2 = a1_2 + t1_2;
1231                        *pc0 = t0;

1233                        break;

1235                case 5:
1236                        j0 = n0 & 1;
1237                        j1 = ( xsb1 + 0x4000 ) & 0xffff8000;
1238                        j2 = n2 & 1;
1239                        HI(&t1) = j1;
1240                        LO(&t1) = 0;
1241                        x0_or_one[0] = x0;
1242                        x0_or_one[2] = -x0;
1243                        x2_or_one[0] = x2;
1244                        x2_or_one[2] = -x2;
1245                        y0_or_zero[0] = y0;
1246                        y0_or_zero[2] = -y0;
1247                        x1 = ( x1 - t1 ) + y1;
1248                        y2_or_zero[0] = y2;
1249                        y2_or_zero[2] = -y2;
1250                        z0 = x0 * x0;
1251                        z1 = x1 * x1;
1252                        z2 = x2 * x2;
1253                        t0 = z0 * ( poly3[j0] + z0 * poly4[j0] );
1254                        t1 = z1 * ( qq1 + z1 * qq2 );
1255                        t2 = z2 * ( poly3[j2] + z2 * poly4[j2] );
1256                        t0 = z0 * ( poly1[j0] + z0 * ( poly2[j0] + t0 ) );
1257                        w1 = x1 * ( one + z1 * ( pp1 + z1 * pp2 ) );
1258                        t2 = z2 * ( poly1[j2] + z2 * ( poly2[j2] + t2 ) );
1259                        j1 = ( ( ( j1 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
1260                        xsb1 = ( xsb1 >> 30 ) & 2;
1261                        n1 ^= ( xsb1 & ~( n1 << 1 ) );
1262                        xsb1 |= 1;

1264                        a1_1 = __vlibm_TBL_sincos_hi[j1+n1];
1265                        a2_1 = __vlibm_TBL_sincos_hi[j1+((n1+xsb1)&3)];

1267                        t0 = x0_or_one[n0] + ( y0_or_zero[n0] + x0_or_one[n0] *
1268                        t2_1 = __vlibm_TBL_sincos_lo[j1+((n1+xsb1)&3)] - ( a1_1*
1269                        t2 = x2_or_one[n2] + ( y2_or_zero[n2] + x2_or_one[n2] *

1271                        *py0 = t0;
1272                        *pc1 = a2_1 + t2_1;
1273                        *py2 = t2;

1275                        n0 = (n0 + 1) & 3;
1276                        n2 = (n2 + 1) & 3;
1277                        j0 = (j0 + 1) & 1;
1278                        j2 = (j2 + 1) & 1;

1280                        t0 = z0 * ( poly3[j0] + z0 * poly4[j0] );
1281                        t1_1 = a2_1*w1 + a1_1*t1;
1282                        t2 = z2 * ( poly3[j2] + z2 * poly4[j2] );

1284                        t0 = z0 * ( poly1[j0] + z0 * ( poly2[j0] + t0 ) );
1285                        t1_1 += __vlibm_TBL_sincos_lo[j1+n1];
1286                        t2 = z2 * ( poly1[j2] + z2 * ( poly2[j2] + t2 ) );

1288                        t0 = x0_or_one[n0] + ( y0_or_zero[n0] + x0_or_one[n0] *
1289                        t2 = x2_or_one[n2] + ( y2_or_zero[n2] + x2_or_one[n2] *

1291                        *pc0 = t0;
1292                        *py1 = a1_1 + t1_1;
```

```
1293                        *pc2 = t2;

1295                        break;

1297                case 6:
1298                        j0 = n0 & 1;
1299                        j1 = n1 & 1;
1300                        j2 = ( xsb2 + 0x4000 ) & 0xffff8000;
1301                        HI(&t2) = j2;
1302                        LO(&t2) = 0;
1303                        x0_or_one[0] = x0;
1304                        x0_or_one[2] = -x0;
1305                        x1_or_one[0] = x1;
1306                        x1_or_one[2] = -x1;
1307                        y0_or_zero[0] = y0;
1308                        y0_or_zero[2] = -y0;
1309                        y1_or_zero[0] = y1;
1310                        y1_or_zero[2] = -y1;
1311                        x2 = ( x2 - t2 ) + y2;
1312                        z0 = x0 * x0;
1313                        z1 = x1 * x1;
1314                        z2 = x2 * x2;
1315                        t0 = z0 * ( poly3[j0] + z0 * poly4[j0] );
1316                        t1 = z1 * ( poly3[j1] + z1 * poly4[j1] );
1317                        t2 = z2 * ( qq1 + z2 * qq2 );
1318                        t0 = z0 * ( poly1[j0] + z0 * ( poly2[j0] + t0 ) );
1319                        t1 = z1 * ( poly1[j1] + z1 * ( poly2[j1] + t1 ) );
1320                        w2 = x2 * ( one + z2 * ( pp1 + z2 * pp2 ) );
1321                        j2 = ( ( ( j2 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
1322                        xsb2 = ( xsb2 >> 30 ) & 2;
1323                        n2 ^= ( xsb2 & ~( n2 << 1 ) );
1324                        xsb2 |= 1;

1326                        a1_2 = __vlibm_TBL_sincos_hi[j2+n2];
1327                        a2_2 = __vlibm_TBL_sincos_hi[j2+((n2+xsb2)&3)];

1329                        t0 = x0_or_one[n0] + ( y0_or_zero[n0] + x0_or_one[n0] *
1330                        t1 = x1_or_one[n1] + ( y1_or_zero[n1] + x1_or_one[n1] *
1331                        t2_2 = __vlibm_TBL_sincos_lo[j2+((n2+xsb2)&3)] - ( a1_2*

1333                        *py0 = t0;
1334                        *py1 = t1;
1335                        *pc2 = a2_2 + t2_2;

1337                        n0 = (n0 + 1) & 3;
1338                        n1 = (n1 + 1) & 3;
1339                        j0 = (j0 + 1) & 1;
1340                        j1 = (j1 + 1) & 1;

1342                        t0 = z0 * ( poly3[j0] + z0 * poly4[j0] );
1343                        t1 = z1 * ( poly3[j1] + z1 * poly4[j1] );
1344                        t1_2 = a2_2*w2 + a1_2*t2;

1346                        t0 = z0 * ( poly1[j0] + z0 * ( poly2[j0] + t0 ) );
1347                        t1 = z1 * ( poly1[j1] + z1 * ( poly2[j1] + t1 ) );
1348                        t1_2 += __vlibm_TBL_sincos_lo[j2+n2];

1350                        t0 = x0_or_one[n0] + ( y0_or_zero[n0] + x0_or_one[n0] *
1351                        t1 = x1_or_one[n1] + ( y1_or_zero[n1] + x1_or_one[n1] *

1353                        *pc0 = t0;
1354                        *pc1 = t1;
1355                        *py2 = a1_2 + t1_2;

1357                        break;
```



```
1359                case 7:
1360                        j0 = n0 & 1;
1361                        j1 = n1 & 1;
1362                        j2 = n2 & 1;
1363                        x0_or_one[0] = x0;
1364                        x0_or_one[2] = -x0;
1365                        x1_or_one[0] = x1;
1366                        x1_or_one[2] = -x1;
1367                        x2_or_one[0] = x2;
1368                        x2_or_one[2] = -x2;
1369                        y0_or_zero[0] = y0;
1370                        y0_or_zero[2] = -y0;
1371                        y1_or_zero[0] = y1;
1372                        y1_or_zero[2] = -y1;
1373                        y2_or_zero[0] = y2;
1374                        y2_or_zero[2] = -y2;
1375                        z0 = x0 * x0;
1376                        z1 = x1 * x1;
1377                        z2 = x2 * x2;
1378                        t0 = z0 * ( poly3[j0] + z0 * poly4[j0] );
1379                        t1 = z1 * ( poly3[j1] + z1 * poly4[j1] );
1380                        t2 = z2 * ( poly3[j2] + z2 * poly4[j2] );
1381                        t0 = z0 * ( poly1[j0] + z0 * ( poly2[j0] + t0 ) );
1382                        t1 = z1 * ( poly1[j1] + z1 * ( poly2[j1] + t1 ) );
1383                        t2 = z2 * ( poly1[j2] + z2 * ( poly2[j2] + t2 ) );
1384                        t0 = x0_or_one[n0] + ( y0_or_zero[n0] + x0_or_one[n0] *
1385                        t1 = x1_or_one[n1] + ( y1_or_zero[n1] + x1_or_one[n1] *
1386                        t2 = x2_or_one[n2] + ( y2_or_zero[n2] + x2_or_one[n2] *
1387                        *py0 = t0;
1388                        *py1 = t1;
1389                        *py2 = t2;

1391                        n0 = (n0 + 1) & 3;
1392                        n1 = (n1 + 1) & 3;
1393                        n2 = (n2 + 1) & 3;
1394                        j0 = (j0 + 1) & 1;
1395                        j1 = (j1 + 1) & 1;
1396                        j2 = (j2 + 1) & 1;
1397                        t0 = z0 * ( poly3[j0] + z0 * poly4[j0] );
1398                        t1 = z1 * ( poly3[j1] + z1 * poly4[j1] );
1399                        t2 = z2 * ( poly3[j2] + z2 * poly4[j2] );
1400                        t0 = z0 * ( poly1[j0] + z0 * ( poly2[j0] + t0 ) );
1401                        t1 = z1 * ( poly1[j1] + z1 * ( poly2[j1] + t1 ) );
1402                        t2 = z2 * ( poly1[j2] + z2 * ( poly2[j2] + t2 ) );
1403                        t0 = x0_or_one[n0] + ( y0_or_zero[n0] + x0_or_one[n0] *
1404                        t1 = x1_or_one[n1] + ( y1_or_zero[n1] + x1_or_one[n1] *
1405                        t2 = x2_or_one[n2] + ( y2_or_zero[n2] + x2_or_one[n2] *
1406                        *pc0 = t0;
1407                        *pc1 = t1;
1408                        *pc2 = t2;
1409                        break;
1410                }

1412                x += stridex;
1413                y += stridey;
1414                c += stridec;
1415                i = 0;
1416        } while ( --n > 0 );

1418        if ( i > 0 )
1419        {
1420                double          a1_0, a1_1, a2_0, a2_1;
1421                double          t0, t1, t1_0, t1_1, t2_0, t2_1;
1422                double          fn0, fn1, a0, a1, w0, w1, y0, y1;
1423                double          z0, z1;
1424                unsigned        j0, j1;
```

```
1425                int             n0, n1;
1427                if ( i > 1 )
1428                {
1429                        n1 = (int) ( x1 * invpio2 + half[xsb1] );
1430                        fn1 = (double) n1;
1431                        n1 &= 3;
1432                        a1 = x1 - fn1 * pio2_1;
1433                        w1 = fn1 * pio2_2;
1434                        x1 = a1 - w1;
1435                        y1 = ( a1 - x1 ) - w1;
1436                        a1 = x1;
1437                        w1 = fn1 * pio2_3 - y1;
1438                        x1 = a1 - w1;
1439                        y1 = ( a1 - x1 ) - w1;
1440                        a1 = x1;
1441                        w1 = fn1 * pio2_3t - y1;
1442                        x1 = a1 - w1;
1443                        y1 = ( a1 - x1 ) - w1;
1444                        xsb1 = HI(&x1);
1445                        if ( ( xsb1 & ~0x80000000 ) < 0x3fc40000 )
1446                        {
1447                                j1 = n1 & 1;
1448                                x1_or_one[0] = x1;
1449                                x1_or_one[2] = -x1;
1450                                y1_or_zero[0] = y1;
1451                                y1_or_zero[2] = -y1;
1452                                z1 = x1 * x1;
1453                                t1 = z1 * ( poly3[j1] + z1 * poly4[j1] );
1454                                t1 = z1 * ( poly1[j1] + z1 * ( poly2[j1] + t1 )
1455                                t1 = x1_or_one[n1] + ( y1_or_zero[n1] + x1_or_on
1456                                *py1 = t1;
1457                                n1 = (n1 + 1) & 3;
1458                                j1 = (j1 + 1) & 1;
1459                                t1 = z1 * ( poly3[j1] + z1 * poly4[j1] );
1460                                t1 = z1 * ( poly1[j1] + z1 * ( poly2[j1] + t1 )
1461                                t1 = x1_or_one[n1] + ( y1_or_zero[n1] + x1_or_on
1462                                *pc1 = t1;
1463                        }
1464                        else
1465                        {
1466                                j1 = ( xsb1 + 0x4000 ) & 0xffff8000;
1467                                HI(&t1) = j1;
1468                                LO(&t1) = 0;
1469                                x1 = ( x1 - t1 ) + y1;
1470                                z1 = x1 * x1;
1471                                t1 = z1 * ( qq1 + z1 * qq2 );
1472                                w1 = x1 * ( one + z1 * ( pp1 + z1 * pp2 ) );
1473                                j1 = ( ( ( j1 & ~0x80000000 ) - 0x3fc40000 ) >>
1474                                xsb1 = ( xsb1 >> 30 ) & 2;
1475                                n1 ^= ( xsb1 & ~( n1 << 1 ) );
1476                                xsb1 |= 1;
1477                                a1_1 = __vlibm_TBL_sincos_hi[j1+n1];
1478                                a2_1 = __vlibm_TBL_sincos_hi[j1+((n1+xsb1)&3)];
1479                                t2_1 = __vlibm_TBL_sincos_lo[j1+((n1+xsb1)&3)] -
1480                                *pc1 = a2_1 + t2_1;
1481                                t1_1 = a2_1*w1 + a1_1*t1;
1482                                t1_1 += __vlibm_TBL_sincos_lo[j1+n1];
1483                                *py1 = a1_1 + t1_1;
1484                        }
1485                }
1486                n0 = (int) ( x0 * invpio2 + half[xsb0] );
1487                fn0 = (double) n0;
1488                n0 &= 3;
1489                a0 = x0 - fn0 * pio2_1;
1490                w0 = fn0 * pio2_2;
```



```
1491                x0 = a0 - w0;
1492                y0 = ( a0 - x0 ) - w0;
1493                a0 = x0;
1494                w0 = fn0 * pio2_3 - y0;
1495                x0 = a0 - w0;
1496                y0 = ( a0 - x0 ) - w0;
1497                a0 = x0;
1498                w0 = fn0 * pio2_3t - y0;
1499                x0 = a0 - w0;
1500                y0 = ( a0 - x0 ) - w0;
1501                xsb0 = HI(&x0);
1502                if ( ( xsb0 & ~0x80000000 ) < 0x3fc40000 )
1503                {
1504                        j0 = n0 & 1;
1505                        x0_or_one[0] = x0;
1506                        x0_or_one[2] = -x0;
1507                        y0_or_zero[0] = y0;
1508                        y0_or_zero[2] = -y0;
1509                        z0 = x0 * x0;
1510                        t0 = z0 * ( poly3[j0] + z0 * poly4[j0] );
1511                        t0 = z0 * ( poly1[j0] + z0 * ( poly2[j0] + t0 ) );
1512                        t0 = x0_or_one[n0] + ( y0_or_zero[n0] + x0_or_one[n0] *
1513                        *py0 = t0;
1514                        n0 = (n0 + 1) & 3;
1515                        j0 = (j0 + 1) & 1;
1516                        t0 = z0 * ( poly3[j0] + z0 * poly4[j0] );
1517                        t0 = z0 * ( poly1[j0] + z0 * ( poly2[j0] + t0 ) );
1518                        t0 = x0_or_one[n0] + ( y0_or_zero[n0] + x0_or_one[n0] *
1519                        *pc0 = t0;
1520                }
1521                else
1522                {
1523                        j0 = ( xsb0 + 0x4000 ) & 0xffff8000;
1524                        HI(&t0) = j0;
1525                        LO(&t0) = 0;
1526                        x0 = ( x0 - t0 ) + y0;
1527                        z0 = x0 * x0;
1528                        t0 = z0 * ( qq1 + z0 * qq2 );
1529                        w0 = x0 * ( one + z0 * ( pp1 + z0 * pp2 ) );
1530                        j0 = ( ( ( j0 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
1531                        xsb0 = ( xsb0 >> 30 ) & 2;
1532                        n0 ^= ( xsb0 & ~( n0 << 1 ) );
1533                        xsb0 |= 1;
1534                        a1_0 = __vlibm_TBL_sincos_hi[j0+n0];
1535                        a2_0 = __vlibm_TBL_sincos_hi[j0+((n0+xsb0)&3)];
1536                        t2_0 = __vlibm_TBL_sincos_lo[j0+((n0+xsb0)&3)] - ( a1_0*
1537                        *pc0 = a2_0 + t2_0;
1538                        t1_0 = a2_0*w0 + a1_0*t0;
1539                        t1_0 += __vlibm_TBL_sincos_lo[j0+n0];
1540                        *py0 = a1_0 + t1_0;
1541                }
1542        }
1544        if ( biguns ) {
1545                __vlibm_vsincos_big( nsave, xsave, sxsave, ysave, sysave, csave,
1546        }
1547 }
```

```
**********************************************************
   45820 Thu Oct  9 19:48:55 2014
new/usr/src/tools/cw/cw.c
rollback tools/cw/cw.c
patch01 - 693 import Sun Devpro Math Library
rollback tools/cw/cw.c
patch01 - 693 import Sun Devpro Math Library
**********************************************************
```

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

/*
 * Copyright 2011, Richard Lowe.
 */
/*
 * Copyright 2010 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

/*
 * Wrapper for the GNU C compiler to make it accept the Sun C compiler
 * arguments where possible.
 *
 * Since the translation is inexact, this is something of a work-in-progress.
 *
 */

/* If you modify this file, you must increment CW_VERSION */
#define CW_VERSION      "1.30"
#define CW_VERSION      "1.29"

/*
 * -#           Verbose mode
 * -###         Show compiler commands built by driver, no compilation
 * -A<name[(tokens)]>   Preprocessor predicate assertion
 * -B<[static|dynamic]> Specify dynamic or static binding
 * -C           Prevent preprocessor from removing comments
 * -c           Compile only - produce .o files, suppress linking
 * -cg92        Alias for -xtarget=ss1000
 * -D<name[=token]>     Associate name with token as if by #define
 * -d[y|n]      dynamic [-dy] or static [-dn] option to linker
 * -E           Compile source through preprocessor only, output to stdout
 * -erroff=<t>  Suppress warnings specified by tags t(%none, %all, <tag list>)
 * -errtags=<a> Display messages with tags a(no, yes)
 * -errwarn=<t> Treats warnings specified by tags t(%none, %all, <tag list>)
 *              as errors
 * -fast        Optimize using a selection of options
 * -fd          Report old-style function definitions and declarations
```



```
 * -features=zla        Allow zero-length arrays
 * -flags       Show this summary of compiler options
 * -fnonstd     Initialize floating-point hardware to non-standard preferences
 * -fns[=<yes|no>] Select non-standard floating point mode
 * -fprecision=<p> Set FP rounding precision mode p(single, double, extended)
 * -fround=<r>  Select the IEEE rounding mode in effect at startup
 * -fsimple[=<n>] Select floating-point optimization preferences <n>
 * -fsingle     Use single-precision arithmetic (-Xt and -Xs modes only)
 * -ftrap=<t>   Select floating-point trapping mode in effect at startup
 * -fstore      force floating pt. values to target precision on assignment
 * -G           Build a dynamic shared library
 * -g           Compile for debugging
 * -H           Print path name of each file included during compilation
 * -h <name>    Assign <name> to generated dynamic shared library
 * -I<dir>      Add <dir> to preprocessor #include file search path
 * -i           Passed to linker to ignore any LD_LIBRARY_PATH setting
 * -keeptmp     Keep temporary files created during compilation
 * -KPIC        Compile position independent code with 32-bit addresses
 * -Kpic        Compile position independent code
 * -L<dir>      Pass to linker to add <dir> to the library search path
 * -l<name>     Link with library lib<name>.a or lib<name>.so
 * -mc          Remove duplicate strings from .comment section of output files
 * -mr          Remove all strings from .comment section of output files
 * -mr,"string" Remove all strings and append "string" to .comment section
 * -mt          Specify options needed when compiling multi-threaded code
 * -native      Find available processor, generate code accordingly
 * -nofstore    Do not force floating pt. values to target precision
 *              on assignment
 * -nolib       Same as -xnolib
 * -noqueue     Disable queuing of compiler license requests
 * -norunpath   Do not build in a runtime path for shared libraries
 * -O           Use default optimization level (-xO2 or -xO3. Check man page.)
 * -o <outputfile> Set name of output file to <outputfile>
 * -P           Compile source through preprocessor only, output to .i  file
 * -PIC         Alias for -KPIC or -xcode=pic32
 * -p           Compile for profiling with prof
 * -pic         Alias for -Kpic or -xcode=pic13
 * -Q[y|n]      Emit/don't emit identification info to output file
 * -qp          Compile for profiling with prof
 * -R<dir[:dir]> Build runtime search path list into executable
 * -S           Compile and only generate assembly code (.s)
 * -s           Strip symbol table from the executable file
 * -t           Turn off duplicate symbol warnings when linking
 * -U<name>     Delete initial definition of preprocessor symbol <name>
 * -V           Report version number of each compilation phase
 * -v           Do stricter semantic checking
 * -W<c>,<arg>  Pass <arg> to specified component <c> (a,l,m,p,0,2,h,i,u)
 * -w           Suppress compiler warning messages
 * -Xa          Compile assuming ANSI C conformance, allow K & R extensions
 *              (default mode)
 * -Xc          Compile assuming strict ANSI C conformance
 * -Xs          Compile assuming (pre-ANSI) K & R C style code
 * -Xt          Compile assuming K & R conformance, allow ANSI C
 * -x386        Generate code for the 80386 processor
 * -x486        Generate code for the 80486 processor
 * -xarch=<a>   Specify target architecture instruction set
 * -xbuiltin[=<b>] When profitable inline, or substitute intrinisic functions
 *              for system functions, b={%all,%none}
 * -xCC         Accept C++ style comments
 * -xchar_byte_order=<o> Specify multi-char byte order <o> (default, high, low)
 * -xchip=<c>   Specify the target processor for use by the optimizer
 * -xcode=<c>   Generate different code for forming addresses
 * -xcrossfile[=<n>] Enable optimization and inlining across source files,
 *              n={0|1}
 * -xe          Perform only syntax/semantic checking, no code generation
 * -xF          Compile for later mapfile reordering or unused section
```

```
 124  *              elimination
 125  * -xhelp=<f>   Display on-line help information f(flags, readme, errors)
 126  * -xildoff     Cancel -xildon
 127  * -xildon      Enable use of the incremental linker, ild
 128  * -xinline=[<a>,...,<a>]  Attempt inlining of specified user routines,
 129  *              <a>={%auto,func,no%func}
 130  * -xlibmieee   Force IEEE 754 return values for math routines in
 131  *              exceptional cases
 132  * -xlibmil     Inline selected libm math routines for optimization
 133  * -xlic_lib=sunperf    Link in the Sun supplied performance libraries
 134  * -xlicinfo    Show license server information
 135  * -xM          Generate makefile dependencies
 136  * -xM1         Generate makefile dependencies, but exclude /usr/include
 137  * -xmaxopt=[off,1,2,3,4,5] maximum optimization level allowed on #pragma opt
 138  * -xnolib      Do not link with default system libraries
 139  * -xnolibmil   Cancel -xlibmil on command line
 140  * -xO<n>       Generate optimized code (n={1|2|3|4|5})
 141  * -xP          Print prototypes for function definitions
 142  * -xpentium    Generate code for the pentium processor
 143  * -xpg         Compile for profiling with gprof
 144  * -xprofile=<p> Collect data for a profile or use a profile to optimize
 145  *              <p>={{collect,use}[:<path>],tcov}
 146  * -xregs=<r>   Control register allocation
 147  * -xs          Allow debugging without object (.o) files
 148  * -xsb         Compile for use with the WorkShop source browser
 149  * -xsbfast     Generate only WorkShop source browser info, no compilation
 150  * -xsfpconst   Represent unsuffixed floating point constants as single
 151  *              precision
 152  * -xspace      Do not do optimizations that increase code size
 153  * -xstrconst   Place string literals into read-only data segment
 154  * -xtarget=<t> Specify target system for optimization
 155  * -xtemp=<dir> Set directory for temporary files to <dir>
 156  * -xtime       Report the execution time for each compilation phase
 157  * -xtransition Emit warnings for differences between K&R C and ANSI C
 158  * -xtrigraphs[=<yes|no>] Enable|disable trigraph translation
 159  * -xunroll=n   Enable unrolling loops n times where possible
 160  * -Y<c>,<dir>  Specify <dir> for location of component <c> (a,l,m,p,0,h,i,u)
 161  * -YA,<dir>    Change default directory searched for components
 162  * -YI,<dir>    Change default directory searched for include files
 163  * -YP,<dir>    Change default directory for finding libraries files
 164  * -YS,<dir>    Change default directory for startup object files
 165  */

 167 /*
 168  * Translation table:
 169  */
 170 /*
 171  * -#                           -v
 172  * -###                         error
 173  * -A<name[(tokens)]>           pass-thru
 174  * -B<[static|dynamic]>         pass-thru (syntax error for anything else)
 175  * -C                           pass-thru
 176  * -c                           pass-thru
 177  * -cg92                        -m32 -mcpu=v8 -mtune=supersparc (SPARC only)
 178  * -D<name[=token]>             pass-thru
 179  * -dy or -dn                   -Wl,-dy or -Wl,-dn
 180  * -E                           pass-thru
 181  * -erroff=E_EMPTY_TRANSLATION_UNIT ignore
 182  * -errtags=%all                -Wall
 183  * -errwarn=%all                -Werror else -Wno-error
 184  * -fast                        error
 185  * -fd                          error
 186  * -features=zla                ignore
 187  * -flags                       --help
 188  * -fnonstd                     error
 189  * -fns[=<yes|no>]              error
```



```
 190  * -fprecision=<p>             error
 191  * -fround=<r>                 error
 192  * -fsimple[=<n>]              error
 193  * -fsingle[=<n>]              error
 194  * -ftrap=<t>                  error
 195  * -fstore                     error
 196  * -G                          pass-thru
 197  * -g                          pass-thru
 198  * -H                          pass-thru
 199  * -h <name>                   pass-thru
 200  * -I<dir>                     pass-thru
 201  * -i                          pass-thru
 202  * -keeptmp                    -save-temps
 203  * -KPIC                       -fPIC
 204  * -Kpic                       -fpic
 205  * -L<dir>                     pass-thru
 206  * -l<name>                    pass-thru
 207  * -mc                         error
 208  * -mr                         error
 209  * -mr,"string"                error
 210  * -mt                         -D_REENTRANT
 211  * -native                     error
 212  * -nofstore                   error
 213  * -nolib                      -nodefaultlibs
 214  * -noqueue                    ignore
 215  * -norunpath                  ignore
 216  * -O                          -O1 (Check the man page to be certain)
 217  * -o <outputfile>             pass-thru
 218  * -P                          -E -o filename.i (or error)
 219  * -PIC                        -fPIC (C++ only)
 220  * -p                          pass-thru
 221  * -pic                        -fpic (C++ only)
 222  * -Q[y|n]                     error
 223  * -qp                         -p
 224  * -R<dir[:dir]>               pass-thru
 225  * -S                          pass-thru
 226  * -s                          -Wl,-s
 227  * -t                          -Wl,-t
 228  * -U<name>                    pass-thru
 229  * -V                          --version
 230  * -v                          -Wall
 231  * -Wa,<arg>                   pass-thru
 232  * -Wp,<arg>                   pass-thru except -xc99=<a>
 233  * -Wl,<arg>                   pass-thru
 234  * -W{m,0,2,h,i,u>             error/ignore
 235  * -Wu,-xmodel=kernel          -ffreestanding -mcmodel=kernel -mno-red-zone
 236  * -xmodel=kernel              -ffreestanding -mcmodel=kernel -mno-red-zone
 237  * -Wu,-save_args              -msave-args
 238  * -w                          pass-thru
 239  * -Xa                         -std=iso9899:199409 or -ansi
 240  * -Xc                         -ansi -pedantic
 241  * -Xt                         error
 242  * -Xs                         -traditional -std=c89
 243  * -x386                       -march=i386 (x86 only)
 244  * -x486                       -march=i486 (x86 only)
 245  * -xarch=<a>                  table
 246  * -xbuiltin[=<b>]             -fbuiltin (-fno-builtin otherwise)
 247  * -xCC                        ignore
 248  * -xchar_byte_order=<o>       error
 249  * -xchip=<c>                  table
 250  * -xcode=<c>                  table
 251  * -xdebugformat=<format>      ignore (always use dwarf-2 for gcc)
 252  * -xcrossfile[=<n>]           ignore
 253  * -xe                         error
 254  * -xF                         error
 255  * -xhelp=<f>                  error
```

```
 256  * -xildoff                    ignore
 257  * -xildon                     ignore
 258  * -xinline                    ignore
 259  * -xlibmieee                  error
 260  * -xlibmil                    error
 261  * -xlic_lib=sunperf           error
 262  * -xM                         -M
 263  * -xM1                        -MM
 264  * -xmaxopt=[...]              error
 265  * -xnolib                     -nodefaultlibs
 266  * -xnolibmil                  error
 267  * -xO<n>                      -O<n>
 268  * -xP                         error
 269  * -xpentium                   -march=pentium (x86 only)
 270  * -xpg                        error
 271  * -xprofile=<p>               error
 272  * -xregs=<r>                  table
 273  * -xs                         error
 274  * -xsb                        error
 275  * -xsbfast                    error
 276  * -xsfpconst                  error
 277  * -xspace                     ignore (-not -Os)
 278  * -xstrconst                  ignore
 279  * -xtarget=<t>                table
 280  * -xtemp=<dir>                error
 281  * -xtime                      error
 282  * -xtransition                -Wtransition
 283  * -xtrigraphs=<yes|no>        -trigraphs -notrigraphs
 284  * -xunroll=n                  error
 285  * -W0,-xdbggen=no%usedonly    -fno-eliminate-unused-debug-symbols
 286  *                             -fno-eliminate-unused-debug-types
 287  * -Y<c>,<dir>                 error
 288  * -YA,<dir>                   error
 289  * -YI,<dir>                   -nostdinc -I<dir>
 290  * -YP,<dir>                   error
 291  * -YS,<dir>                   error
 292  */

 294 #include <stdio.h>
 295 #include <sys/types.h>
 296 #include <unistd.h>
 297 #include <string.h>
 298 #include <stdlib.h>
 299 #include <ctype.h>
 300 #include <fcntl.h>
 301 #include <errno.h>
 302 #include <stdarg.h>
 303 #include <sys/utsname.h>
 304 #include <sys/param.h>
 305 #include <sys/isa_defs.h>
 306 #include <sys/wait.h>
 307 #include <sys/stat.h>

 309 #define CW_F_CXX        0x01
 310 #define CW_F_SHADOW     0x02
 311 #define CW_F_EXEC       0x04
 312 #define CW_F_ECHO       0x08
 313 #define CW_F_XLATE      0x10
 314 #define CW_F_PROG       0x20

 316 typedef enum cw_compiler {
 317         CW_C_CC = 0,
 318         CW_C_GCC
 319 } cw_compiler_t;
```

unchanged_portion_omitted_


```
 388 /*
 389  * The translation table for the -xarch= flag used in the Studio compilers.
 390  */
 391 static const xarch_table_t xtbl[] = {
 392 #if defined(__x86)
 393         { "generic",    SS11 },
 394         { "generic64",  (SS11|M64), { "-m64", "-mtune=opteron" } },
 395         { "amd64",      (SS11|M64), { "-m64", "-mtune=opteron" } },
 396         { "386",        SS11,   { "-march=i386" } },
 397         { "pentium_pro", SS11,  { "-march=pentiumpro" } },
 398         { "sse",        SS11, { "-msse", "-mfpmath=sse" } },
 399         { "sse2",       SS11, { "-msse2", "-mfpmath=sse" } },
 400 #elif defined(__sparc)
 401         { "generic",    (SS11|M32), { "-m32", "-mcpu=v8" } },
 402         { "generic64",  (SS11|M64), { "-m64", "-mcpu=v9" } },
 403         { "v8",         (SS11|M32), { "-m32", "-mcpu=v8", "-mno-v8plus" } },
 404         { "v8plus",     (SS11|M32), { "-m32", "-mcpu=v9", "-mv8plus" } },
 405         { "v8plusa",    (SS11|M32), { "-m32", "-mcpu=ultrasparc", "-mv8plus",
 406                         "-mvis" } },
 407         { "v8plusb",    (SS11|M32), { "-m32", "-mcpu=ultrasparc3", "-mv8plus",
 408                         "-mvis" } },
 409         { "v9",         (SS11|M64), { "-m64", "-mcpu=v9" } },
 410         { "v9a",        (SS11|M64), { "-m64", "-mcpu=ultrasparc", "-mvis" } },
 411         { "v9b",        (SS11|M64), { "-m64", "-mcpu=ultrasparc3", "-mvis" } },
 412         { "sparc",      SS12, { "-mcpu=v9", "-mv8plus" } },
 413         { "sparcvis",   SS12, { "-mcpu=ultrasparc", "-mvis" } },
 414         { "sparcvis2",  SS12, { "-mcpu=ultrasparc3", "-mvis" } }
 415 #endif
 416 };
```

unchanged_portion_omitted_

```
**********************************************************
    4224 Thu Oct  9 19:48:56 2014
new/usr/src/uts/common/sys/ccompile.h
Revert "remove unused v from libmvec"
This reverts commit e853d278ee4b7f2a8c2117cc598cfc68b4e3f29b.
fix system-library-math.mf
update libm manifests
14071:dece9aafe99a - fix build problems on sparc
remove unused v from libmvec
fix for patch09 - use __GNU_UNUSED - fix cstyle
fix for patch09 - use __GNU_UNUSED
Revert "remove unused v from libmvec"
This reverts commit e853d278ee4b7f2a8c2117cc598cfc68b4e3f29b.
fix system-library-math.mf
update libm manifests
14071:dece9aafe99a - fix build problems on sparc
remove unused v from libmvec
fix for patch09 - use __GNU_UNUSED - fix cstyle
fix for patch09 - use __GNU_UNUSED
**********************************************************
```

```
  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, Version 1.0 only
  6  * (the "License").  You may not use this file except in compliance
  7  * with the License.
  8  *
  9  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 10  * or http://www.opensolaris.org/os/licensing.
 11  * See the License for the specific language governing permissions
 12  * and limitations under the License.
 13  *
 14  * When distributing Covered Code, include this CDDL HEADER in each
 15  * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 16  * If applicable, add the following below this CDDL HEADER, with the
 17  * fields enclosed by brackets "[]" replaced with your own identifying
 18  * information: Portions Copyright [yyyy] [name of copyright owner]
 19  *
 20  * CDDL HEADER END
 21  */
 22 /*
 23  * Copyright 2004 Sun Microsystems, Inc.  All rights reserved.
 24  * Use is subject to license terms.
 25  */

 27 #ifndef _SYS_CCOMPILE_H
 28 #define _SYS_CCOMPILE_H

 30 /*
 31  * This file contains definitions designed to enable different compilers
 32  * to be used harmoniously on Solaris systems.
 33  */

 35 #ifdef  __cplusplus
 36 extern "C" {
 37 #endif

 39 /*
 40  * Allow for version tests for compiler bugs and features.
 41  */
 42 #if defined(__GNUC__)
 43 #define __GNUC_VERSION  \
 44         (__GNUC__ * 10000 + __GNUC_MINOR__ * 100 + __GNUC_PATCHLEVEL__)
 45 #else
 46 #define __GNUC_VERSION  0
```



```
 47 #endif

 49 #if defined(__ATTRIBUTE_IMPLEMENTED) || defined(__GNUC__)

 51 /*
 52  * analogous to lint's PRINTFLIKEn
 53  */
 54 #define __sun_attr___PRINTFLIKE__(__n)  \
 55                 __attribute__((__format__(printf, __n, (__n)+1)))
 56 #define __sun_attr___VPRINTFLIKE__(__n) \
 57                 __attribute__((__format__(printf, __n, 0)))

 59 /*
 60  * Handle the kernel printf routines that can take '%b' too
 61  */
 62 #if __GNUC_VERSION < 30402
 63 /*
 64  * XX64 at least this doesn't work correctly yet with 3.4.1 anyway!
 65  */
 66 #define __sun_attr___KPRINTFLIKE__      __sun_attr___PRINTFLIKE__
 67 #define __sun_attr___KVPRINTFLIKE__     __sun_attr___VPRINTFLIKE__
 68 #else
 69 #define __sun_attr___KPRINTFLIKE__(__n) \
 70                 __attribute__((__format__(cmn_err, __n, (__n)+1)))
 71 #define __sun_attr___KVPRINTFLIKE__(__n) \
 72                 __attribute__((__format__(cmn_err, __n, 0)))
 73 #endif

 75 /*
 76  * This one's pretty obvious -- the function never returns
 77  */
 78 #define __sun_attr___noreturn__ __attribute__((__noreturn__))

 80 /*
 81  * The function is 'extern inline' and expects GNU C89 behaviour, not C99
 82  * behaviour.
 83  *
 84  * Should only be used on 'extern inline' definitions for GCC.
 85  */
 86 #if __GNUC_VERSION >= 40200
 87 #define __sun_attr___gnu_inline__       __attribute__((__gnu_inline__))
 88 #else
 89 #define __sun_attr___gnu_inline__
 90 #endif

 92 /*
 93  * The function has control flow such that it may return multiple times (in
 94  * the manner of setjmp or vfork)
 95  */
 96 #if __GNUC_VERSION >= 40100
 97 #define __sun_attr___returns_twice__    __attribute__((__returns_twice__))
 98 #else
 99 #define __sun_attr___returns_twice__
100 #endif

102 /*
103  * This is an appropriate label for functions that do not
104  * modify their arguments, e.g. strlen()
105  */
106 #define __sun_attr___pure__     __attribute__((__pure__))

108 /*
109  * This is a stronger form of __pure__. Can be used for functions
110  * that do not modify their arguments and don't depend on global
111  * memory.
112  */
```

```
113 #define __sun_attr___const__    __attribute__((__const__))

115 /*
116  * structure packing like #pragma pack(1)
117  */
118 #define __sun_attr___packed__   __attribute__((__packed__))

120 #define ___sun_attr_inner(__a)  __sun_attr_##__a
121 #define __sun_attr__(__a)       ___sun_attr_inner __a

123 #else   /* __ATTRIBUTE_IMPLEMENTED || __GNUC__ */

125 #define __sun_attr__(__a)

127 #endif  /* __ATTRIBUTE_IMPLEMENTED || __GNUC__ */

129 /*
130  * Shorthand versions for readability
131  */

133 #define __PRINTFLIKE(__n)       __sun_attr__((__PRINTFLIKE__(__n)))
134 #define __VPRINTFLIKE(__n)      __sun_attr__((__VPRINTFLIKE__(__n)))
135 #define __KPRINTFLIKE(__n)      __sun_attr__((__KPRINTFLIKE__(__n)))
136 #define __KVPRINTFLIKE(__n)     __sun_attr__((__KVPRINTFLIKE__(__n)))
137 #define __NORETURN              __sun_attr__((__noreturn__))
138 #define __GNU_INLINE            __inline__ __sun_attr__((__gnu_inline__))
139 #define __RETURNS_TWICE         __sun_attr__((__returns_twice__))
140 #define __CONST                 __sun_attr__((__const__))
141 #define __PURE                  __sun_attr__((__pure__))
142 #define __GNU_UNUSED    __attribute__((__unused__))

144 #ifdef  __cplusplus
145 }
```

*unchanged_portion_omitted_*