```
**********************************************************
   19733 Sun May 11 12:15:21 2014
new/usr/src/lib/libm/Makefile.com
**********************************************************
#
# 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      = \
                  __libx_errno.o

COBJS_sparc     = \
                  $(COBJS_i386) \
                  _TBL_atan.o \
                  _TBL_exp2.o \
                  _TBL_log.o \
                  _TBL_log2.o \
                  _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 CERRWARN        += -_gcc=-Wno-switch
515 CERRWARN        += -_gcc=-Wno-uninitialized
515 CERRWARN        += -_gcc=-Wno-parentheses
516 CERRWARN        += -_gcc=-Wno-unused-variable

518 CPPFLAGS        += -DLIBM_BUILD

520 CFLAGS          += $(C_BIGPICFLAGS)
521 CFLAGS64        += $(C_BIGPICFLAGS)

523 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 \
        ../common/R/sincosf.c \
        ../common/R/sincospif.c \
        ../common/R/sqrtf.c \
        ../common/R/tanf.c \
```

```
656         ../common/R/tanhf.c

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



```
722         ../common/Q/sinhl.c \
723         ../common/Q/sinl.c \
724         ../common/Q/sinpil.c \
725         ../common/Q/sqrtl.c \
726         ../common/Q/tanhl.c \
727         ../common/Q/tanl.c

729 SRCS_Q_sparc = \
730         $(SRCS_Q)

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

```
788         ../common/complex/csinhl.c \
789         ../common/complex/csinl.c \
790         ../common/complex/csqrt.c \
791         ../common/complex/csqrtf.c \
792         ../common/complex/csqrtl.c \
793         ../common/complex/ctan.c \
794         ../common/complex/ctanf.c \
795         ../common/complex/ctanh.c \
796         ../common/complex/ctanhf.c \
797         ../common/complex/ctanhl.c \
798         ../common/complex/ctanl.c \
799         ../common/complex/k_atan2.c \
800         ../common/complex/k_atan2l.c \
801         ../common/complex/k_cexp.c \
802         ../common/complex/k_cexpl.c \
803         ../common/complex/k_clog_r.c \
804         ../common/complex/k_clog_rl.c

806 SRCS_m9x_i386 = \
807         ../common/m9x/__fex_sse.c \
808         ../common/m9x/feprec.c \
809         ../common/m9x/__fex_i386.c

811 SRCS_m9x_i386_i386 = \
812         ../common/m9x/lroundf.c

814 SRCS_m9x_i386_amd64 = \
815         ../common/m9x/llrint.c \
816         ../common/m9x/llrintf.c \
817         ../common/m9x/llrintl.c \
818         ../common/m9x/nexttowardl.c \
819         ../common/m9x/remquo.c \
820         ../common/m9x/remquof.c \
821         ../common/m9x/round.c \
822         ../common/m9x/roundl.c \
823         ../common/m9x/scalbln.c \
824         ../common/m9x/scalblnf.c \
825         ../common/m9x/scalblnl.c \
826         ../common/m9x/trunc.c \
827         ../common/m9x/truncl.c

829 # sparc
830 SRCS_m9x_sparc_sparc = \
831         ../common/m9x/lrint.c \
832         ../common/m9x/lrintf.c \
833         ../common/m9x/lrintl.c \
834         ../common/m9x/lround.c \
835         ../common/m9x/lroundf.c \
836         ../common/m9x/lroundl.c

838 SRCS_m9x_sparc = \
839         ../common/m9x/__fex_sparc.c \
840         ../common/m9x/llrint.c \
841         ../common/m9x/llrintf.c \
842         ../common/m9x/llrintl.c \
843         ../common/m9x/nexttowardl.c \
844         ../common/m9x/remquo.c \
845         ../common/m9x/remquof.c \
846         ../common/m9x/remquol.c \
847         ../common/m9x/round.c \
848         ../common/m9x/roundl.c \
849         ../common/m9x/scalbln.c \
850         ../common/m9x/scalblnf.c \
851         ../common/m9x/scalblnl.c \
852         ../common/m9x/trunc.c \
853         ../common/m9x/truncl.c
```



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

904 SRCS_C_sparc = \
905         ../common/C/__tan.c \
906         ../common/C/_TBL_atan.c \
907         ../common/C/_TBL_exp2.c \
908         ../common/C/_TBL_log.c \
909         ../common/C/_TBL_log2.c \
910         ../common/C/_TBL_tan.c \
911         ../common/C/acos.c \
912         ../common/C/asin.c \
913         ../common/C/atan.c \
914         ../common/C/atan2.c \
915         ../common/C/ceil.c \
916         ../common/C/cos.c \
917         ../common/C/exp.c \
918         ../common/C/exp10.c \
919         ../common/C/exp2.c \
```

```
920         ../common/C/expm1.c \
921         ../common/C/floor.c \
922         ../common/C/fmod.c \
923         ../common/C/hypot.c \
924         ../common/C/ilogb.c \
925         ../common/C/isnan.c \
926         ../common/C/log.c \
927         ../common/C/log10.c \
928         ../common/C/log2.c \
929         ../common/C/pow.c \
930         ../common/C/remainder.c \
931         ../common/C/rint.c \
932         ../common/C/scalbn.c \
933         ../common/C/sin.c \
934         ../common/C/sincos.c \
935         ../common/C/tan.c

937 SRCS_i386_i386  = \
938         ../common/C/__libx_errno.c

940 SRCS_sparc_sparc = \
941         $(SRCS_i386_i386)

943 SRCS_sparc_sparcv9 = \
944         ../common/C/copysign.c \
945         ../common/C/fabs.c \
946         ../common/C/nextafter.c

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

982 SRCS_C = \
983         $(SRCS_C_$(MACH)) \
984         $(SRCS_C_i386_$(TARGET_ARCH)) \
985         ../common/C/__cos.c \
```



```
986         ../common/C/__lgamma.c \
987         ../common/C/__rem_pio2.c \
988         ../common/C/__rem_pio2m.c \
989         ../common/C/__sin.c \
990         ../common/C/__sincos.c \
991         ../common/C/__xpg6.c \
992         ../common/C/_lib_version.c \
993         ../common/C/_SVID_error.c \
994         ../common/C/_TBL_ipio2.c \
995         ../common/C/_TBL_sin.c \
996         ../common/C/acosh.c \
997         ../common/C/asinh.c \
998         ../common/C/atan2pi.c \
999         ../common/C/atanh.c \
1000        ../common/C/cbrt.c \
1001        ../common/C/cosh.c \
1002        ../common/C/erf.c \
1003        ../common/C/gamma.c \
1004        ../common/C/gamma_r.c \
1005        ../common/C/j0.c \
1006        ../common/C/j1.c \
1007        ../common/C/jn.c \
1008        ../common/C/lgamma.c \
1009        ../common/C/lgamma_r.c \
1010        ../common/C/log1p.c \
1011        ../common/C/logb.c \
1012        ../common/C/matherr.c \
1013        ../common/C/scalb.c \
1014        ../common/C/signgam.c \
1015        ../common/C/significand.c \
1016        ../common/C/sincospi.c \
1017        ../common/C/sinh.c \
1018        ../common/C/sqrt.c \
1019        ../common/C/tanh.c

1021 SRCS    = \
1022        $(SRCS_Q_$(MACH)) \
1023        $(SRCS_LD_$(MACH)) \
1024        $(SRCS_R) \
1025        $(SRCS_complex) \
1026        $(SRCS_C)

1028 .KEEP_STATE:

1030 all:    $(LIBS)

1032 lint:   lintcheck
```

****
3962 Sun May 11 12:15:23 2014
new/usr/src/lib/libm/amd64/src/libm_inlines.h
****
_____unchanged_portion_omitted_

```
 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)
 140             : "+x" (d1), "+x" (tmpd)
 141             : "x" (d2), "r" (0x7fffffffffffffff));

 143         return (d1);
 144 }
```

_____unchanged_portion_omitted_

****
22354 Sun May 11 12:15:24 2014
new/usr/src/lib/libm/common/C/_SVID_error.c
****
_____unchanged_portion_omitted_

```
114 #define NaN     C[0].d
115 #define PI_RZ   C[1].d

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

122 double
123 _SVID_libm_err(double x, double y, int type) {
124         struct exception        exc;
125         double                  t, w, ieee_retval = 0;
125         double                  t, w, ieee_retval;
126         enum version            lib_version = _lib_version;
127         int                     iy;

129         /* force libm_ieee behavior in SUSv3 mode */
130         if ((__xpg6 & _C99SUSv3_math_errexcept) != 0)
131                 lib_version = libm_ieee;
132         if (lib_version == c_issue_4) {
133                 (void) fflush(stdout);
134         }
135         exc.arg1 = x;
136         exc.arg2 = y;
137         switch (type) {
138         case 1:
139                 /* acos(|x|>1) */
140                 exc.type = DOMAIN;
141                 exc.name = "acos";
142                 ieee_retval = setexception(3, 1.0);
143                 exc.retval = 0.0;
144                 if (lib_version == strict_ansi) {
145                         errno = EDOM;
146                 } else if (!matherr(&exc)) {
147                         if (lib_version == c_issue_4) {
148                                 (void) write(2, "acos: DOMAIN error\n", 19);
149                         }
150                         errno = EDOM;
151                 }
152                 break;
153         case 2:
154                 /* asin(|x|>1) */
155                 exc.type = DOMAIN;
156                 exc.name = "asin";
157                 exc.retval = 0.0;
158                 ieee_retval = setexception(3, 1.0);
159                 if (lib_version == strict_ansi) {
160                         errno = EDOM;
161                 } else if (!matherr(&exc)) {
162                         if (lib_version == c_issue_4) {
163                                 (void) write(2, "asin: DOMAIN error\n", 19);
164                         }
165                         errno = EDOM;
166                 }
167                 break;
168         case 3:
169                 /* atan2(+-0,+-0) */
170                 exc.arg1 = y;
171                 exc.arg2 = x;
172                 exc.type = DOMAIN;
```



```
173                 exc.name = "atan2";
174                 ieee_retval = copysign(1.0, x) == 1.0 ? y :
175                         copysign(PI_RZ + DBL_MIN, y);
176                 exc.retval = 0.0;
177                 if (lib_version == strict_ansi) {
178                         errno = EDOM;
179                 } else if (!matherr(&exc)) {
180                         if (lib_version == c_issue_4) {
181                                 (void) write(2, "atan2: DOMAIN error\n", 20);
182                         }
183                         errno = EDOM;
184                 }
185                 break;
186         case 4:
187                 /* hypot(finite,finite) overflow */
188                 exc.type = OVERFLOW;
189                 exc.name = "hypot";
190                 ieee_retval = Inf;
191                 if (lib_version == c_issue_4)
192                         exc.retval = HUGE;
193                 else
194                         exc.retval = HUGE_VAL;
195                 if (lib_version == strict_ansi)
196                         errno = ERANGE;
197                 else if (!matherr(&exc))
198                         errno = ERANGE;
199                 break;
200         case 5:
201                 /* cosh(finite) overflow */
202                 exc.type = OVERFLOW;
203                 exc.name = "cosh";
204                 ieee_retval = setexception(2, 1.0);
205                 if (lib_version == c_issue_4)
206                         exc.retval = HUGE;
207                 else
208                         exc.retval = HUGE_VAL;
209                 if (lib_version == strict_ansi)
210                         errno = ERANGE;
211                 else if (!matherr(&exc))
212                         errno = ERANGE;
213                 break;
214         case 6:
215                 /* exp(finite) overflow */
216                 exc.type = OVERFLOW;
217                 exc.name = "exp";
218                 ieee_retval = setexception(2, 1.0);
219                 if (lib_version == c_issue_4)
220                         exc.retval = HUGE;
221                 else
222                         exc.retval = HUGE_VAL;
223                 if (lib_version == strict_ansi)
224                         errno = ERANGE;
225                 else if (!matherr(&exc))
226                         errno = ERANGE;
227                 break;
228         case 7:
229                 /* exp(finite) underflow */
230                 exc.type = UNDERFLOW;
231                 exc.name = "exp";
232                 ieee_retval = setexception(1, 1.0);
233                 exc.retval = 0.0;
234                 if (lib_version == strict_ansi)
235                         errno = ERANGE;
236                 else if (!matherr(&exc))
237                         errno = ERANGE;
238                 break;
```

```
239         case 8:
240                 /* y0(0) = -inf */
241                 exc.type = DOMAIN;      /* should be SING for IEEE */
242                 exc.name = "y0";
243                 ieee_retval = setexception(0, -1.0);
244                 if (lib_version == c_issue_4)
245                         exc.retval = -HUGE;
246                 else
247                         exc.retval = -HUGE_VAL;
248                 if (lib_version == strict_ansi) {
249                         errno = EDOM;
250                 } else if (!matherr(&exc)) {
251                         if (lib_version == c_issue_4) {
252                                 (void) write(2, "y0: DOMAIN error\n", 17);
253                         }
254                         errno = EDOM;
255                 }
256                 break;
257         case 9:
258                 /* y0(x<0) = NaN */
259                 exc.type = DOMAIN;
260                 exc.name = "y0";
261                 ieee_retval = setexception(3, 1.0);
262                 if (lib_version == c_issue_4)
263                         exc.retval = -HUGE;
264                 else
265                         exc.retval = -HUGE_VAL;
266                 if (lib_version == strict_ansi) {
267                         errno = EDOM;
268                 } else if (!matherr(&exc)) {
269                         if (lib_version == c_issue_4) {
270                                 (void) write(2, "y0: DOMAIN error\n", 17);
271                         }
272                         errno = EDOM;
273                 }
274                 break;
275         case 10:
276                 /* y1(0) = -inf */
277                 exc.type = DOMAIN;      /* should be SING for IEEE */
278                 exc.name = "y1";
279                 ieee_retval = setexception(0, -1.0);
280                 if (lib_version == c_issue_4)
281                         exc.retval = -HUGE;
282                 else
283                         exc.retval = -HUGE_VAL;
284                 if (lib_version == strict_ansi) {
285                         errno = EDOM;
286                 } else if (!matherr(&exc)) {
287                         if (lib_version == c_issue_4) {
288                                 (void) write(2, "y1: DOMAIN error\n", 17);
289                         }
290                         errno = EDOM;
291                 }
292                 break;
293         case 11:
294                 /* y1(x<0) = NaN */
295                 exc.type = DOMAIN;
296                 exc.name = "y1";
297                 ieee_retval = setexception(3, 1.0);
298                 if (lib_version == c_issue_4)
299                         exc.retval = -HUGE;
300                 else
301                         exc.retval = -HUGE_VAL;
302                 if (lib_version == strict_ansi) {
303                         errno = EDOM;
304                 } else if (!matherr(&exc)) {
```



```
305                         if (lib_version == c_issue_4) {
306                                 (void) write(2, "y1: DOMAIN error\n", 17);
307                         }
308                         errno = EDOM;
309                 }
310                 break;
311         case 12:
312                 /* yn(n,0) = -inf */
313                 exc.type = DOMAIN;      /* should be SING for IEEE */
314                 exc.name = "yn";
315                 ieee_retval = setexception(0, -1.0);
316                 if (lib_version == c_issue_4)
317                         exc.retval = -HUGE;
318                 else
319                         exc.retval = -HUGE_VAL;
320                 if (lib_version == strict_ansi) {
321                         errno = EDOM;
322                 } else if (!matherr(&exc)) {
323                         if (lib_version == c_issue_4) {
324                                 (void) write(2, "yn: DOMAIN error\n", 17);
325                         }
326                         errno = EDOM;
327                 }
328                 break;
329         case 13:
330                 /* yn(x<0) = NaN */
331                 exc.type = DOMAIN;
332                 exc.name = "yn";
333                 ieee_retval = setexception(3, 1.0);
334                 if (lib_version == c_issue_4)
335                         exc.retval = -HUGE;
336                 else
337                         exc.retval = -HUGE_VAL;
338                 if (lib_version == strict_ansi) {
339                         errno = EDOM;
340                 } else if (!matherr(&exc)) {
341                         if (lib_version == c_issue_4) {
342                                 (void) write(2, "yn: DOMAIN error\n", 17);
343                         }
344                         errno = EDOM;
345                 }
346                 break;
347         case 14:
348                 /* lgamma(finite) overflow */
349                 exc.type = OVERFLOW;
350                 exc.name = "lgamma";
351                 ieee_retval = setexception(2, 1.0);
352                 if (lib_version == c_issue_4)
353                         exc.retval = HUGE;
354                 else
355                         exc.retval = HUGE_VAL;
356                 if (lib_version == strict_ansi)
357                         errno = ERANGE;
358                 else if (!matherr(&exc))
359                         errno = ERANGE;
360                 break;
361         case 15:
362                 /* lgamma(-integer) or lgamma(0) */
363                 exc.type = SING;
364                 exc.name = "lgamma";
365                 ieee_retval = setexception(0, 1.0);
366                 if (lib_version == c_issue_4)
367                         exc.retval = HUGE;
368                 else
369                         exc.retval = HUGE_VAL;
370                 if (lib_version == strict_ansi) {
```

```
371                     errno = EDOM;
372             } else if (!matherr(&exc)) {
373                     if (lib_version == c_issue_4) {
374                             (void) write(2, "lgamma: SING error\n", 19);
375                     }
376                     errno = EDOM;
377             }
378             break;
379     case 16:
380             /* log(0) */
381             exc.type = SING;
382             exc.name = "log";
383             ieee_retval = setexception(0, -1.0);
384             if (lib_version == c_issue_4)
385                     exc.retval = -HUGE;
386             else
387                     exc.retval = -HUGE_VAL;
388             if (lib_version == strict_ansi) {
389                     errno = ERANGE;
390             } else if (!matherr(&exc)) {
391                     if (lib_version == c_issue_4) {
392                             (void) write(2, "log: SING error\n", 16);
393                             errno = EDOM;
394                     } else {
395                             errno = ERANGE;
396                     }
397             }
398             break;
399     case 17:
400             /* log(x<0) */
401             exc.type = DOMAIN;
402             exc.name = "log";
403             ieee_retval = setexception(3, 1.0);
404             if (lib_version == c_issue_4)
405                     exc.retval = -HUGE;
406             else
407                     exc.retval = -HUGE_VAL;
408             if (lib_version == strict_ansi) {
409                     errno = EDOM;
410             } else if (!matherr(&exc)) {
411                     if (lib_version == c_issue_4) {
412                             (void) write(2, "log: DOMAIN error\n", 18);
413                     }
414                     errno = EDOM;
415             }
416             break;
417     case 18:
418             /* log10(0) */
419             exc.type = SING;
420             exc.name = "log10";
421             ieee_retval = setexception(0, -1.0);
422             if (lib_version == c_issue_4)
423                     exc.retval = -HUGE;
424             else
425                     exc.retval = -HUGE_VAL;
426             if (lib_version == strict_ansi) {
427                     errno = ERANGE;
428             } else if (!matherr(&exc)) {
429                     if (lib_version == c_issue_4) {
430                             (void) write(2, "log10: SING error\n", 18);
431                             errno = EDOM;
432                     } else {
433                             errno = ERANGE;
434                     }
435             }
436             break;
```



```
437     case 19:
438             /* log10(x<0) */
439             exc.type = DOMAIN;
440             exc.name = "log10";
441             ieee_retval = setexception(3, 1.0);
442             if (lib_version == c_issue_4)
443                     exc.retval = -HUGE;
444             else
445                     exc.retval = -HUGE_VAL;
446             if (lib_version == strict_ansi) {
447                     errno = EDOM;
448             } else if (!matherr(&exc)) {
449                     if (lib_version == c_issue_4) {
450                             (void) write(2, "log10: DOMAIN error\n", 20);
451                     }
452                     errno = EDOM;
453             }
454             break;
455     case 20:
456             /* pow(0.0,0.0) */
457             /* error only if lib_version == c_issue_4 */
458             exc.type = DOMAIN;
459             exc.name = "pow";
460             exc.retval = 0.0;
461             ieee_retval = 1.0;
462             if (lib_version != c_issue_4) {
463                     exc.retval = 1.0;
464             } else if (!matherr(&exc)) {
465                     (void) write(2, "pow(0,0): DOMAIN error\n", 23);
466                     errno = EDOM;
467             }
468             break;
469     case 21:
470             /* pow(x,y) overflow */
471             exc.type = OVERFLOW;
472             exc.name = "pow";
473             exc.retval = (lib_version == c_issue_4)? HUGE : HUGE_VAL;
474             if (signbit(x)) {
475                     t = rint(y);
476                     if (t == y) {
477                             w = rint(0.5 * y);
478                             if (t != w + w) {       /* y is odd */
479                                     exc.retval = -exc.retval;
480                             }
481                     }
482             }
483             ieee_retval = setexception(2, exc.retval);
484             if (lib_version == strict_ansi)
485                     errno = ERANGE;
486             else if (!matherr(&exc))
487                     errno = ERANGE;
488             break;
489     case 22:
490             /* pow(x,y) underflow */
491             exc.type = UNDERFLOW;
492             exc.name = "pow";
493             exc.retval = 0.0;
494             if (signbit(x)) {
495                     t = rint(y);
496                     if (t == y) {
497                             w = rint(0.5 * y);
498                             if (t != w + w) /* y is odd */
499                                     exc.retval = -exc.retval;
500                     }
501             }
502             ieee_retval = setexception(1, exc.retval);
```

```
503                 if (lib_version == strict_ansi)
504                         errno = ERANGE;
505                 else if (!matherr(&exc))
506                         errno = ERANGE;
507                 break;
508         case 23:
509                 /* (+-0)**neg */
510                 exc.type = DOMAIN;
511                 exc.name = "pow";
512                 ieee_retval = setexception(0, 1.0);
513                 {
514                         int ahy, k, j, yisint, ly, hx;
515                         /* INDENT OFF */
516                         /*
517                          * determine if y is an odd int when x = -0
518                          * yisint = 0       ... y is not an integer
519                          * yisint = 1       ... y is an odd int
520                          * yisint = 2       ... y is an even int
521                          */
522                         /* INDENT ON */
523                         hx  = __HI(x);
524                         ahy = __HI(y)&0x7fffffff;
525                         ly  = __LO(y);

527                         yisint = 0;
528                         if (ahy >= 0x43400000) {
529                                 yisint = 2;     /* even integer y */
530                         } else if (ahy >= 0x3ff00000) {
531                                 k = (ahy >> 20) - 0x3ff;        /* exponent */
532                                 if (k > 20) {
533                                         j = ly >> (52 - k);
534                                         if ((j << (52 - k)) == ly)
535                                                 yisint = 2 - (j & 1);
536                                 } else if (ly == 0) {
537                                         j = ahy >> (20 - k);
538                                         if ((j << (20 - k)) == ahy)
539                                                 yisint = 2 - (j & 1);
540                                 }
541                         }
542                         if (hx < 0 && yisint == 1)
543                                 ieee_retval = -ieee_retval;
544                 }
545                 if (lib_version == c_issue_4)
546                         exc.retval = 0.0;
547                 else
548                         exc.retval = -HUGE_VAL;
549                 if (lib_version == strict_ansi) {
550                         errno = EDOM;
551                 } else if (!matherr(&exc)) {
552                         if (lib_version == c_issue_4) {
553                                 (void) write(2, "pow(0,neg): DOMAIN error\n",
554                                     25);
555                         }
556                         errno = EDOM;
557                 }
558                 break;
559         case 24:
560                 /* neg**non-integral */
561                 exc.type = DOMAIN;
562                 exc.name = "pow";
563                 ieee_retval = setexception(3, 1.0);
564                 if (lib_version == c_issue_4)
565                         exc.retval = 0.0;
566                 else
567                         exc.retval = ieee_retval;       /* X/Open allow NaN */
568                 if (lib_version == strict_ansi) {
```



```
569                         errno = EDOM;
570                 } else if (!matherr(&exc)) {
571                         if (lib_version == c_issue_4) {
572                                 (void) write(2,
573                                     "neg**non-integral: DOMAIN error\n", 32);
574                         }
575                         errno = EDOM;
576                 }
577                 break;
578         case 25:
579                 /* sinh(finite) overflow */
580                 exc.type = OVERFLOW;
581                 exc.name = "sinh";
582                 ieee_retval = copysign(Inf, x);
583                 if (lib_version == c_issue_4)
584                         exc.retval = x > 0.0 ? HUGE : -HUGE;
585                 else
586                         exc.retval = x > 0.0 ? HUGE_VAL : -HUGE_VAL;
587                 if (lib_version == strict_ansi)
588                         errno = ERANGE;
589                 else if (!matherr(&exc))
590                         errno = ERANGE;
591                 break;
592         case 26:
593                 /* sqrt(x<0) */
594                 exc.type = DOMAIN;
595                 exc.name = "sqrt";
596                 ieee_retval = setexception(3, 1.0);
597                 if (lib_version == c_issue_4)
598                         exc.retval = 0.0;
599                 else
600                         exc.retval = ieee_retval;       /* quiet NaN */
601                 if (lib_version == strict_ansi) {
602                         errno = EDOM;
603                 } else if (!matherr(&exc)) {
604                         if (lib_version == c_issue_4) {
605                                 (void) write(2, "sqrt: DOMAIN error\n", 19);
606                         }
607                         errno = EDOM;
608                 }
609                 break;
610         case 27:
611                 /* fmod(x,0) */
612                 exc.type = DOMAIN;
613                 exc.name = "fmod";
614                 if (fp_class(x) == fp_quiet)
615                         ieee_retval = NaN;
616                 else
617                         ieee_retval = setexception(3, 1.0);
618                 if (lib_version == c_issue_4)
619                         exc.retval = x;
620                 else
621                         exc.retval = ieee_retval;
622                 if (lib_version == strict_ansi) {
623                         errno = EDOM;
624                 } else if (!matherr(&exc)) {
625                         if (lib_version == c_issue_4) {
626                                 (void) write(2, "fmod:  DOMAIN error\n", 20);
627                         }
628                         errno = EDOM;
629                 }
630                 break;
631         case 28:
632                 /* remainder(x,0) */
633                 exc.type = DOMAIN;
634                 exc.name = "remainder";
```

```
635                 if (fp_class(x) == fp_quiet)
636                         ieee_retval = NaN;
637                 else
638                         ieee_retval = setexception(3, 1.0);
639                 exc.retval = NaN;
640                 if (lib_version == strict_ansi) {
641                         errno = EDOM;
642                 } else if (!matherr(&exc)) {
643                         if (lib_version == c_issue_4) {
644                                 (void) write(2, "remainder: DOMAIN error\n",
645                                     24);
646                         }
647                         errno = EDOM;
648                 }
649                 break;
650         case 29:
651                 /* acosh(x<1) */
652                 exc.type = DOMAIN;
653                 exc.name = "acosh";
654                 ieee_retval = setexception(3, 1.0);
655                 exc.retval = NaN;
656                 if (lib_version == strict_ansi) {
657                         errno = EDOM;
658                 } else if (!matherr(&exc)) {
659                         if (lib_version == c_issue_4) {
660                                 (void) write(2, "acosh: DOMAIN error\n", 20);
661                         }
662                         errno = EDOM;
663                 }
664                 break;
665         case 30:
666                 /* atanh(|x|>1) */
667                 exc.type = DOMAIN;
668                 exc.name = "atanh";
669                 ieee_retval = setexception(3, 1.0);
670                 exc.retval = NaN;
671                 if (lib_version == strict_ansi) {
672                         errno = EDOM;
673                 } else if (!matherr(&exc)) {
674                         if (lib_version == c_issue_4) {
675                                 (void) write(2, "atanh: DOMAIN error\n", 20);
676                         }
677                         errno = EDOM;
678                 }
679                 break;
680         case 31:
681                 /* atanh(|x|=1) */
682                 exc.type = SING;
683                 exc.name = "atanh";
684                 ieee_retval = setexception(0, x);
685                 exc.retval = ieee_retval;
686                 if (lib_version == strict_ansi) {
687                         errno = ERANGE;
688                 } else if (!matherr(&exc)) {
689                         if (lib_version == c_issue_4) {
690                                 (void) write(2, "atanh: SING error\n", 18);
691                                 errno = EDOM;
692                         } else {
693                                 errno = ERANGE;
694                         }
695                 }
696                 break;
697         case 32:
698                 /* scalb overflow; SVID also returns +-HUGE_VAL */
699                 exc.type = OVERFLOW;
700                 exc.name = "scalb";
```



```
701                 ieee_retval = setexception(2, x);
702                 exc.retval = x > 0.0 ? HUGE_VAL : -HUGE_VAL;
703                 if (lib_version == strict_ansi)
704                         errno = ERANGE;
705                 else if (!matherr(&exc))
706                         errno = ERANGE;
707                 break;
708         case 33:
709                 /* scalb underflow */
710                 exc.type = UNDERFLOW;
711                 exc.name = "scalb";
712                 ieee_retval = setexception(1, x);
713                 exc.retval = ieee_retval;       /* +-0.0 */
714                 if (lib_version == strict_ansi)
715                         errno = ERANGE;
716                 else if (!matherr(&exc))
717                         errno = ERANGE;
718                 break;
719         case 34:
720                 /* j0(|x|>X_TLOSS) */
721                 exc.type = TLOSS;
722                 exc.name = "j0";
723                 exc.retval = 0.0;
724                 ieee_retval = y;
725                 if (lib_version == strict_ansi) {
726                         errno = ERANGE;
727                 } else if (!matherr(&exc)) {
728                         if (lib_version == c_issue_4) {
729                                 (void) write(2, exc.name, 2);
730                                 (void) write(2, ": TLOSS error\n", 14);
731                         }
732                         errno = ERANGE;
733                 }
734                 break;
735         case 35:
736                 /* y0(x>X_TLOSS) */
737                 exc.type = TLOSS;
738                 exc.name = "y0";
739                 exc.retval = 0.0;
740                 ieee_retval = y;
741                 if (lib_version == strict_ansi) {
742                         errno = ERANGE;
743                 } else if (!matherr(&exc)) {
744                         if (lib_version == c_issue_4) {
745                                 (void) write(2, exc.name, 2);
746                                 (void) write(2, ": TLOSS error\n", 14);
747                         }
748                         errno = ERANGE;
749                 }
750                 break;
751         case 36:
752                 /* j1(|x|>X_TLOSS) */
753                 exc.type = TLOSS;
754                 exc.name = "j1";
755                 exc.retval = 0.0;
756                 ieee_retval = y;
757                 if (lib_version == strict_ansi) {
758                         errno = ERANGE;
759                 } else if (!matherr(&exc)) {
760                         if (lib_version == c_issue_4) {
761                                 (void) write(2, exc.name, 2);
762                                 (void) write(2, ": TLOSS error\n", 14);
763                         }
764                         errno = ERANGE;
765                 }
766                 break;
```

```
767         case 37:
768                 /* y1(x>X_TLOSS) */
769                 exc.type = TLOSS;
770                 exc.name = "y1";
771                 exc.retval = 0.0;
772                 ieee_retval = y;
773                 if (lib_version == strict_ansi) {
774                         errno = ERANGE;
775                 } else if (!matherr(&exc)) {
776                         if (lib_version == c_issue_4) {
777                                 (void) write(2, exc.name, 2);
778                                 (void) write(2, ": TLOSS error\n", 14);
779                         }
780                         errno = ERANGE;
781                 }
782                 break;
783         case 38:
784                 /* jn(|x|>X_TLOSS) */
785                 /* incorrect ieee value: ieee should never be here */
786                 exc.type = TLOSS;
787                 exc.name = "jn";
788                 exc.retval = 0.0;
789                 ieee_retval = 0.0;      /* shall not be used */
790                 if (lib_version == strict_ansi) {
791                         errno = ERANGE;
792                 } else if (!matherr(&exc)) {
793                         if (lib_version == c_issue_4) {
794                                 (void) write(2, exc.name, 2);
795                                 (void) write(2, ": TLOSS error\n", 14);
796                         }
797                         errno = ERANGE;
798                 }
799                 break;
800         case 39:
801                 /* yn(x>X_TLOSS) */
802                 /* incorrect ieee value: ieee should never be here */
803                 exc.type = TLOSS;
804                 exc.name = "yn";
805                 exc.retval = 0.0;
806                 ieee_retval = 0.0;      /* shall not be used */
807                 if (lib_version == strict_ansi) {
808                         errno = ERANGE;
809                 } else if (!matherr(&exc)) {
810                         if (lib_version == c_issue_4) {
811                                 (void) write(2, exc.name, 2);
812                                 (void) write(2, ": TLOSS error\n", 14);
813                         }
814                         errno = ERANGE;
815                 }
816                 break;
817         case 40:
818                 /* gamma(finite) overflow */
819                 exc.type = OVERFLOW;
820                 exc.name = "gamma";
821                 ieee_retval = setexception(2, 1.0);
822                 if (lib_version == c_issue_4)
823                         exc.retval = HUGE;
824                 else
825                         exc.retval = HUGE_VAL;
826                 if (lib_version == strict_ansi)
827                         errno = ERANGE;
828                 else if (!matherr(&exc))
829                         errno = ERANGE;
830                 break;
831         case 41:
832                 /* gamma(-integer) or gamma(0) */
```



```
833                 exc.type = SING;
834                 exc.name = "gamma";
835                 ieee_retval = setexception(0, 1.0);
836                 if (lib_version == c_issue_4)
837                         exc.retval = HUGE;
838                 else
839                         exc.retval = HUGE_VAL;
840                 if (lib_version == strict_ansi) {
841                         errno = EDOM;
842                 } else if (!matherr(&exc)) {
843                         if (lib_version == c_issue_4) {
844                                 (void) write(2, "gamma: SING error\n", 18);
845                         }
846                         errno = EDOM;
847                 }
848                 break;
849         case 42:
850                 /* pow(NaN,0.0) */
851                 /* error if lib_version == c_issue_4 or ansi_1 */
852                 exc.type = DOMAIN;
853                 exc.name = "pow";
854                 exc.retval = x;
855                 ieee_retval = 1.0;
856                 if (lib_version == strict_ansi) {
857                         exc.retval = 1.0;
858                 } else if (!matherr(&exc)) {
859                         if ((lib_version == c_issue_4) || (lib_version == ansi_1
859                         switch (lib_version) {
860                         case c_issue_4:
861                         case ansi_1:
860                                 errno = EDOM;
863                         }
861                 }
862                 break;
863         case 43:
864                 /* log1p(-1) */
865                 exc.type = SING;
866                 exc.name = "log1p";
867                 ieee_retval = setexception(0, -1.0);
868                 if (lib_version == c_issue_4)
869                         exc.retval = -HUGE;
870                 else
871                         exc.retval = -HUGE_VAL;
872                 if (lib_version == strict_ansi) {
873                         errno = ERANGE;
874                 } else if (!matherr(&exc)) {
875                         if (lib_version == c_issue_4) {
876                                 (void) write(2, "log1p: SING error\n", 18);
877                                 errno = EDOM;
878                         } else {
879                                 errno = ERANGE;
880                         }
881                 }
882                 break;
883         case 44:
884                 /* log1p(x<-1) */
885                 exc.type = DOMAIN;
886                 exc.name = "log1p";
887                 ieee_retval = setexception(3, 1.0);
888                 exc.retval = ieee_retval;
889                 if (lib_version == strict_ansi) {
890                         errno = EDOM;
891                 } else if (!matherr(&exc)) {
892                         if (lib_version == c_issue_4) {
893                                 (void) write(2, "log1p: DOMAIN error\n", 20);
894                         }
```

```
895                         errno = EDOM;
896                 }
897                 break;
898         case 45:
899                 /* logb(0) */
900                 exc.type = DOMAIN;
901                 exc.name = "logb";
902                 ieee_retval = setexception(0, -1.0);
903                 exc.retval = -HUGE_VAL;
904                 if (lib_version == strict_ansi)
905                         errno = EDOM;
906                 else if (!matherr(&exc))
907                         errno = EDOM;
908                 break;
909         case 46:
910                 /* nextafter overflow */
911                 exc.type = OVERFLOW;
912                 exc.name = "nextafter";
913                 /*
914                  * The value as returned by setexception is +/-DBL_MAX in
915                  * round-to-{zero,-/+Inf} mode respectively, which is not
916                  * usable.
917                  */
918                 (void) setexception(2, x);
919                 ieee_retval = x > 0 ? Inf : -Inf;
920                 exc.retval = x > 0 ? HUGE_VAL : -HUGE_VAL;
921                 if (lib_version == strict_ansi)
922                         errno = ERANGE;
923                 else if (!matherr(&exc))
924                         errno = ERANGE;
925                 break;
926         case 47:
927                 /* scalb(x,inf) */
928                 iy = ((int *)&y)[HIWORD];
929                 if (lib_version == c_issue_4)
930                         /* SVID3: ERANGE in all cases */
931                         errno = ERANGE;
932                 else if ((x == 0.0 && iy > 0) || (!finite(x) && iy < 0))
933                         /* EDOM for scalb(0,+inf) or scalb(inf,-inf) */
934                         errno = EDOM;
935                 exc.retval = ieee_retval = ((iy < 0)? x / -y : x * y);
936                 break;
937         }
938         switch (lib_version) {
939         case c_issue_4:
940         case ansi_1:
941         case strict_ansi:
942                 return (exc.retval);
943                 /* NOTREACHED */
944         default:
945                 return (ieee_retval);
946         }
947         /* NOTREACHED */
948 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
    5678 Sun May 11 12:15:26 2014
new/usr/src/lib/libm/common/C/__tan.c
**********************************************************
______unchanged_portion_omitted_


#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;
        double a, t, z, w, 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 (ix < 0x3fc40000) {
                if (ix < 0x3e400000) {
                        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);
        }
}
______unchanged_portion_omitted_
```

```
**********************************************************
    4870 Sun May 11 12:15:28 2014
new/usr/src/lib/libm/common/C/asin.c
**********************************************************
______unchanged_portion_omitted_
  88 #define one     xxx[0]
  89 #define huge    xxx[1]
  90 #define pio2_hi xxx[2]
  91 #define pio2_lo xxx[3]
  92 #define pio4_hi xxx[4]
  93 #define pS0     xxx[5]
  94 #define pS1     xxx[6]
  95 #define pS2     xxx[7]
  96 #define pS3     xxx[8]
  97 #define pS4     xxx[9]
  98 #define pS5     xxx[10]
  99 #define qS1     xxx[11]
 100 #define qS2     xxx[12]
 101 #define qS3     xxx[13]
 102 #define qS4     xxx[14]
 103 /* INDENT ON */

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

 110         hx = ((int *) &x)[HIWORD];
 111         ix = hx & 0x7fffffff;
 112         if (ix >= 0x3ff00000) { /* |x| >= 1 */
 113                 if (((ix - 0x3ff00000) | ((int *) &x)[LOWORD]) == 0)
 114                         /* asin(1)=+-pi/2 with inexact */
 115                         return x * pio2_hi + x * pio2_lo;
 116                 else if (isnan(x))
 117 #if defined(FPADD_TRAPS_INCOMPLETE_ON_NAN)
 118                         return ix >= 0x7ff80000 ? x : (x - x) / (x - x);
 119                         /* assumes sparc-like QNaN */
 120 #else
 121                         return (x - x) / (x - x);       /* asin(|x|>1) is NaN */
 122 #endif
 123                 else
 124                         return _SVID_libm_err(x, x, 2);
 125         }
 126         else if (ix < 0x3fe00000) {     /* |x| < 0.5 */
 127                 if (ix < 0x3e400000) {  /* if |x| < 2**-27 */
 128                         if ((i = (int) x) == 0)
 128                         if (huge + x > one)
 129                                 return x;       /* return x with inexact if
 130                                                  * x != 0 */
 131                 }
 132                 else
 133                 t = x * x;
 134                 p = t * (pS0 + t * (pS1 + t * (pS2 + t * (pS3 +
 135                         t * (pS4 + t * pS5)))));
 136                 q = one + t * (qS1 + t * (qS2 + t * (qS3 + t * qS4)));
 137                 w = p / q;
 138                 return x + x * w;
 139         }
 140         /* 1 > |x| >= 0.5 */
 141         w = one - fabs(x);
 142         t = w * 0.5;
 143         p = t * (pS0 + t * (pS1 + t * (pS2 + t * (pS3 + t * (pS4 + t * pS5)))));
 144         q = one + t * (qS1 + t * (qS2 + t * (qS3 + t * qS4)));
 145         s = sqrt(t);
 146         if (ix >= 0x3FEF3333) { /* if |x| > 0.975 */
```



```
 146                 w = p / q;
 147                 t = pio2_hi - (2.0 * (s + s * w) - pio2_lo);
 148         }
 149         else {
 150                 w = s;
 151                 ((int *) &w)[LOWORD] = 0;
 152                 c = (t - w * w) / (s + w);
 153                 r = p / q;
 154                 p = 2.0 * s * r - (pio2_lo - 2.0 * c);
 155                 q = pio4_hi - 2.0 * w;
 156                 t = pio4_hi - (p - q);
 157         }
 158         return hx > 0 ? t : -t;
 159 }
______unchanged_portion_omitted_
```

```
**********************************************************
    8501 Sun May 11 12:15:29 2014
new/usr/src/lib/libm/common/C/expm1.c
**********************************************************
______unchanged_portion_omitted_
 148 #define one             xxx[0]
 149 #define huge            xxx[1]
 150 #define tiny            xxx[2]
 151 #define o_threshold     xxx[3]
 152 #define ln2_hi          xxx[4]
 153 #define ln2_lo          xxx[5]
 154 #define invln2          xxx[6]
 155 #define Q1              xxx[7]
 156 #define Q2              xxx[8]
 157 #define Q3              xxx[9]
 158 #define Q4              xxx[10]
 159 #define Q5              xxx[11]

 161 double
 162 expm1(double x) {
 163         double y, hi, lo, c = 0.0L, t, e, hxs, hfx, r1;
 163         double y, hi, lo, c, t, e, hxs, hfx, r1;
 164         int k, xsb;
 165         unsigned hx;

 167         hx = ((unsigned *) &x)[HIWORD];         /* high word of x */
 168         xsb = hx & 0x80000000;                  /* sign bit of x */
 169         if (xsb == 0)
 170                 y = x;
 171         else
 172                 y = -x;                         /* y = |x| */
 173         hx &= 0x7fffffff;                       /* high word of |x| */

 175         /* filter out huge and non-finite argument */
 176         /* for example exp(38)-1 is approximately 3.1855932e+16 */
 177         if (hx >= 0x4043687A) {                 /* if |x|>=56*ln2  (~38.8162...)
 178                 if (hx >= 0x40862E42) {         /* if |x|>=709.78... -> inf */
 175         /* filter out huge and non-finite arugment */
 176         if (hx >= 0x4043687A) {                 /* if |x|>=56*ln2 */
 177                 if (hx >= 0x40862E42) {         /* if |x|>=709.78... */
 179                         if (hx >= 0x7ff00000) {
 180                                 if (((hx & 0xfffff) | ((int *) &x)[LOWORD])
 181                                         != 0)
 182                                         return x * x;   /* + -> * for Cheetah */
 183                                 else
 184                                         return xsb == 0 ? x : -1.0;     /* exp(+
 185                         }
 186                         if (x > o_threshold)
 187                                 return huge * huge;     /* overflow */
 188                 }
 189                 if (xsb != 0) {         /* x < -56*ln2, return -1.0 w/inexact */
 190                         if (x + tiny < 0.0)             /* raise inexact */
 191                                 return tiny - one;      /* return -1 */
 192                 }
 193         }

 195         /* argument reduction */
 196         if (hx > 0x3fd62e42) {                  /* if  |x| > 0.5 ln2 */
 197                 if (hx < 0x3FF0A2B2) {          /* and |x| < 1.5 ln2 */
 198                         if (xsb == 0) {         /* positive number */
 197                         if (xsb == 0) {
 199                                 hi = x - ln2_hi;
 200                                 lo = ln2_lo;
 201                                 k = 1;
 202                         }
 203                         else { /* negative number */
```



```
 202                         else {
 204                                 hi = x + ln2_hi;
 205                                 lo = -ln2_lo;
 206                                 k = -1;
 207                         }
 208                 }
 209                 else {  /* |x| > 1.5 ln2 */
 208                 else {
 210                         k = (int) (invln2 * x + (xsb == 0 ? 0.5 : -0.5));
 211                         t = k;
 212                         hi = x - t * ln2_hi;    /* t*ln2_hi is exact here */
 213                         lo = t * ln2_lo;
 214                 }
 215                 x = hi - lo;
 216                 c = (hi - x) - lo; /* still at |x| > 0.5 ln2 */
 215                 c = (hi - x) - lo;
 217         }
 218         else if (hx < 0x3c900000) {             /* when |x|<2**-54, return x */
 219                 t = huge + x;           /* return x w/inexact when x != 0 */
 220                 return x - (t - (huge + x));
 221         }
 222         else    /* |x| <= 0.5 ln2 */
 221         else
 223                 k = 0;

 225         /* x is now in primary range */
 226         hfx = 0.5 * x;
 227         hxs = x * hfx;
 228         r1 = one + hxs * (Q1 + hxs * (Q2 + hxs * (Q3 + hxs * (Q4 + hxs * Q5))));
 229         t = 3.0 - r1 * hfx;
 230         e = hxs * ((r1 - t) / (6.0 - x * t));
 231         if (k == 0) /* |x| <= 0.5 ln2 */
 232                 return x - (x * e - hxs);
 233         else {      /* |x| > 0.5 ln2 */
 230         if (k == 0)
 231                 return x - (x * e - hxs);       /* c is 0 */
 232         else {
 234                 e = (x * (e - c) - c);
 235                 e -= hxs;
 236                 if (k == -1)
 237                         return 0.5 * (x - e) - 0.5;
 238                 if (k == 1) {
 237                 if (k == 1)
 239                         if (x < -0.25)
 240                                 return -2.0 * (e - (x + 0.5));
 241                         else
 242                                 return one + 2.0 * (x - e);
 243                 }
 244 #endif /* ! codereview */
 245                 if (k <= -2 || k > 56) {        /* suffice to return exp(x)-1 */
 246                         y = one - (e - x);
 247                         ((int *) &y)[HIWORD] += k << 20;
 248                         return y - one;
 249                 }
 250                 t = one;
 251                 if (k < 20) {
 252                         ((int *) &t)[HIWORD] = 0x3ff00000 - (0x200000 >> k);
 253                                                         /* t = 1 - 2^-k */
 254                         y = t - (e - x);
 255                         ((int *) &y)[HIWORD] += k << 20;
 256                 }
 257                 else {
 258                         ((int *) &t)[HIWORD] = (0x3ff - k) << 20; /* 2^-k */
 259                         y = x - (e + t);
 260                         y += one;
 261                         ((int *) &y)[HIWORD] += k << 20;
```

```
262                 }
263         }
264         return y;
265 }
```

```
**********************************************************
    7265 Sun May 11 12:15:31 2014
new/usr/src/lib/libm/common/C/jn.c
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" 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 = 5.64189583547756286948079451560772585844le-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 a, b, temp;
        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
                                           1    -s-c            -c+s
                                           2    -s+c            -c-s
                                           3     s+c             c-s
                                 */
                switch(n&3) {
                    case 0: temp =  cos(x)+sin(x); break;
                    case 1: temp = -cos(x)+sin(x); break;
                    case 2: temp = -cos(x)-sin(x); break;
                    case 3: temp =  cos(x)-sin(x); break;
                }
                b = invsqrtpi*temp/sqrt(x);
            } else {
                a = j0(x);
                b = j1(x);
                for(i=1;i<n;i++){
                    temp = b;
                    b = b*((GENERIC)(i+i)/x) - a; /* avoid underflow */
                    a = temp;
```

```
128                 }
129             }
130         } else {
131             if(x<1e-9) {        /* use J(n,x) = 1/n!*(x/2)^n */
132                 b = pow(0.5*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                 /* use backward recurrence */
139                 /*                       x      x^2      x^2
140                  *  J(n,x)/J(n-1,x) =  ----   ------   ------   .....
141                  *                       2n  - 2(n+1) - 2(n+2)
142                  *
143                  *                       1      1        1
144                  *  (for large x)   =  ----  ------   ------   .....
145                  *                       2n   2(n+1)   2(n+2)
146                  *                      -- - ------ - ------ -
147                  *                       x     x         x
148                  *
149                  * Let w = 2n/x and h=2/x, then the above quotient
150                  * is equal to the continued fraction:
151                  *                   1
152                  *      = -----------------------
153                  *                  1
154                  *         w - -----------------
155                  *                     1
156                  *              w+h - ---------
157                  *                    w+2h - ...
158                  *
159                  * To determine how many terms needed, let
160                  * Q(0) = w, Q(1) = w(w+h) - 1,
161                  * Q(k) = (w+k*h)*Q(k-1) - Q(k-2),
162                  * When Q(k) > 1e4      good for single
163                  * When Q(k) > 1e9      good for double
164                  * When Q(k) > 1e17     good for quaduple
165                  */
166             /* determin k */
167                 GENERIC t,v;
168                 double q0,q1,h,tmp; int k,m;
169                 w  = (n+n)/(double)x; h = 2.0/(double)x;
170                 q0 = w;  z = w+h; q1 = w*z - 1.0; k=1;
171                 while(q1<1.0e9) {
172                         k += 1; z += h;
173                         tmp = z*q1 - q0;
174                         q0 = q1;
175                         q1 = tmp;
176                 }
177                 m = n+n;
178                 for(t=zero, i = 2*(n+k); i>=m; i -= 2) t = one/(i/x-t);
179                 a = t;
180                 b = one;
181                 /* estimate log((2/x)^n*n!) = n*log(2/x)+n*ln(n)
182                    hence, if n*(log(2n/x)) > ...
183                     single 8.8722839355e+01
184                     double 7.09782712893383973096e+02
185                     long double 1.13565234062941439494919310779707650060170e+04
186                     then recurrent value may overflow and the result is
187                     likely underflow to zero
188                  */
189                 tmp = n;
190                 v = two/x;
191                 tmp = tmp*log(fabs(v*tmp));
192                 if(tmp<7.09782712893383973096e+02) {
193                     for(i=n-1;i>0;i--){
```



```
194                         temp = b;
195                         b = ((i+i)/x)*b - a;
196                         a = temp;
197                     }
198                 } else {
199                     for(i=n-1;i>0;i--){
200                         temp = b;
201                         b = ((i+i)/x)*b - a;
202                         a = temp;
203                         if(b>1e100) {
204                             a /= b;
205                             t /= b;
206                             b  = 1.0;
207                         }
208                     }
209                 }
210                 b = (t*j0(x)/b);
211             }
212         }
213         if(sgn==1) return -b; else return b;
214 }

216 GENERIC
217 yn(int n, GENERIC x) {
218         int i;
219         int sign;
220         GENERIC a, b, temp = 0, ox, on;
220         GENERIC a, b, temp, ox, on;

222         ox = x; on = (GENERIC)n;
223         if(isnan(x)) return x*x;        /* + -> * for Cheetah */
224         if (x <= zero) {
225                 if(x==zero) {
224         if (x <= zero)
225                 if(x==zero)
226                         /* return -one/zero; */
227                         return _SVID_libm_err((GENERIC)n,x,12);
228                 } else {
228                 else
229                         /* return zero/zero; */
230                         return _SVID_libm_err((GENERIC)n,x,13);
231                 }
232         }
233 #endif /* ! codereview */
234         if (!((int) _lib_version == libm_ieee ||
235                 (__xpg6 & _C99SUSv3_math_errexcept) != 0)) {
236             if(x > X_TLOSS) return _SVID_libm_err(on,ox,39);
237         }
238         sign = 1;
239         if(n<0){
240                 n = -n;
241                 if((n&1) == 1) sign = -1;
242         }
243         if(n==0) return(y0(x));
244         if(n==1) return(sign*y1(x));
245         if(!finite(x)) return zero;

247         if(x>1.0e91) {  /* x >> n**2
248                                         Jn(x) = cos(x-(2n+1)*pi/4)*sqrt(2/x*pi)
249                                         Yn(x) = sin(x-(2n+1)*pi/4)*sqrt(2/x*pi)
250                                         Let s=sin(x), c=cos(x),
251                                             xn=x-(2n+1)*pi/4, sqt2 = sqrt(2),then

253                                                n    sin(xn)*sqt2    cos(xn)*sqt2
254                                             ----------------------------------
255                                                0     s-c             c+s
```

```
256                                         1    -s-c            -c+s
257                                         2    -s+c            -c-s
258                                         3     s+c             c-s
259                                 */
260                 switch(n&3) {
261                     case 0: temp =  sin(x)-cos(x); break;
262                     case 1: temp = -sin(x)-cos(x); break;
263                     case 2: temp = -sin(x)+cos(x); break;
264                     case 3: temp =  sin(x)+cos(x); break;
265                 }
266                 b = invsqrtpi*temp/sqrt(x);
267         } else {
268                 a = y0(x);
269                 b = y1(x);
270                 /*
271                  * fix 1262058 and take care of non-default rounding
272                  */
273                 for (i = 1; i < n; i++) {
274                         temp = b;
275                         b *= (GENERIC) (i + i) / x;
276                         if (b <= -DBL_MAX)
277                                 break;
278                         b -= a;
279                         a = temp;
280                 }
281         }
282         if(sign>0) return b; else return -b;
283 }
```

```
**********************************************************
    6359 Sun May 11 12:15:32 2014
new/usr/src/lib/libm/common/C/log1p.c
**********************************************************
______unchanged_portion_omitted_
```

```
 112 #define ln2_hi  xxx[0]
 113 #define ln2_lo  xxx[1]
 114 #define two54   xxx[2]
 115 #define Lp1     xxx[3]
 116 #define Lp2     xxx[4]
 117 #define Lp3     xxx[5]
 118 #define Lp4     xxx[6]
 119 #define Lp5     xxx[7]
 120 #define Lp6     xxx[8]
 121 #define Lp7     xxx[9]
 122 #define zero    xxx[10]

 124 double
 125 log1p(double x) {
 126         double  hfsq, f, c = 0.0, s, z, R, u;
 126         double  hfsq, f, c, s, z, R, u;
 127         int     k, hx, hu, ax;

 129         hx = ((int *)&x)[HIWORD];               /* high word of x */
 130         ax = hx & 0x7fffffff;

 132         if (ax >= 0x7ff00000) { /* x is inf or nan */
 133                 if (((hx - 0xfff00000) | ((int *)&x)[LOWORD]) == 0) /* -inf */
 134                         return (_SVID_libm_err(x, x, 44));
 135                 return (x * x);
 136         }

 138         k = 1;
 139         if (hx < 0x3FDA827A) {  /* x < 0.41422  */
 140                 if (ax >= 0x3ff00000)   /* x <= -1.0 */
 141                         return (_SVID_libm_err(x, x, x == -1.0 ? 43 : 44));
 142                 if (ax < 0x3e200000) {  /* |x| < 2**-29 */
 143                         if (two54 + x > zero && /* raise inexact */
 144                             ax < 0x3c900000)    /* |x| < 2**-54 */
 145                                 return (x);
 146                         else
 147                                 return (x - x * x * 0.5);
 148                 }
 149                 if (hx > 0 || hx <= (int)0xbfd2bec3) {  /* -0.2929<x<0.41422 */
 150                         k = 0;
 151                         f = x;
 152                         hu = 1;
 153                 }
 154         }
 155         /* We will initialize 'c' here. */
 156 #endif /* ! codereview */
 157         if (k != 0) {
 158                 if (hx < 0x43400000) {
 159                         u = 1.0 + x;
 160                         hu = ((int *)&u)[HIWORD];       /* high word of u */
 161                         k = (hu >> 20) - 1023;
 162                         /*
 163                          * correction term
 164                          */
 165                         c = k > 0 ? 1.0 - (u - x) : x - (u - 1.0);
 166                         c /= u;
 167                 } else {
 168                         u = x;
 169                         hu = ((int *)&u)[HIWORD];       /* high word of u */
 170                         k = (hu >> 20) - 1023;
 171                         c = 0;
```



```
 172                 }
 173                 hu &= 0x000fffff;
 174                 if (hu < 0x6a09e) {     /* normalize u */
 175                         ((int *)&u)[HIWORD] = hu | 0x3ff00000;
 176                 } else {                        /* normalize u/2 */
 177                         k += 1;
 178                         ((int *)&u)[HIWORD] = hu | 0x3fe00000;
 179                         hu = (0x00100000 - hu) >> 2;
 180                 }
 181                 f = u - 1.0;
 182         }
 183         hfsq = 0.5 * f * f;
 184         if (hu == 0) {          /* |f| < 2**-20 */
 185                 if (f == zero) {
 186                         if (k == 0)
 187                                 return (zero);
 188                         /* We already initialized 'c' before, when (k != 0) */
 189 #endif /* ! codereview */
 190                         c += k * ln2_lo;
 191                         return (k * ln2_hi + c);
 192                 }
 193                 R = hfsq * (1.0 - 0.66666666666666666 * f);
 194                 if (k == 0)
 195                         return (f - R);
 196                 return (k * ln2_hi - ((R - (k * ln2_lo + c)) - f));
 197         }
 198         s = f / (2.0 + f);
 199         z = s * s;
 200         R = z * (Lp1 + z * (Lp2 + z * (Lp3 + z * (Lp4 + z * (Lp5 +
 201                 z * (Lp6 + z * Lp7))))));
 202         if (k == 0)
 203                 return (f - (hfsq - s * (hfsq + R)));
 204         return (k * ln2_hi - ((hfsq - (s * (hfsq + R) +
 205                 (k * ln2_lo + c))) - f));
 206 }
```

```
**********************************************************
    2274 Sun May 11 12:15:33 2014
new/usr/src/lib/libm/common/C/nextafter.c
**********************************************************
```

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

#include "libm.h"
#include <float.h>              /* DBL_MIN */

double
nextafter(double x, double y) {
        int             hx, hy, k;
        double          ans;
        unsigned        lx;
        volatile double dummy;
#endif /* ! codereview */

        hx = ((int *)&x)[HIWORD];
        lx = ((int *)&x)[LOWORD];
        hy = ((int *)&y)[HIWORD];
        k = (hx & ~0x80000000) | lx;

        if (x == y)
                return (y);             /* C99 requirement */
        if (x != x || y != y)
                return (x * y);
        if (k == 0) {                   /* x = 0 */
                k = hy & 0x80000000;
                ((int *)&ans)[HIWORD] = k;
                ((int *)&ans)[LOWORD] = 1;
        } else if (hx >= 0) {
                if (x > y) {
                        ((int *)&ans)[LOWORD] = lx - 1;
                        k = (lx == 0)? hx - 1 : hx;
                        ((int *)&ans)[HIWORD] = k;
                } else {
                        ((int *)&ans)[LOWORD] = lx + 1;
```



```
                        k  = (lx == 0xffffffff)? hx + 1 : hx;
                        ((int *)&ans)[HIWORD] = k;
                }
        } else {
                if (x < y) {
                        ((int *)&ans)[LOWORD] = lx - 1;
                        k = (lx == 0)? hx - 1 : hx;
                        ((int *)&ans)[HIWORD] = k;
                } else {
                        ((int *)&ans)[LOWORD] = lx + 1;
                        k  = (lx == 0xffffffff)? hx + 1 : hx;
                        ((int *)&ans)[HIWORD] = k;
                }
        }
        k = (k >> 20) & 0x7ff;
        if (k == 0x7ff) {
                /* overflow */
                return (_SVID_libm_err(x, y, 46));
#if !defined(__lint)
        } else if (k == 0) {
                /* underflow */
                dummy = DBL_MIN * copysign(DBL_MIN, x);
                volatile double dummy = DBL_MIN * copysign(DBL_MIN, x);
#endif
        }
        return (ans);
}
______unchanged_portion_omitted_
```

```
**********************************************************
   10202 Sun May 11 12:15:35 2014
new/usr/src/lib/libm/common/C/pow.c
**********************************************************
______unchanged_portion_omitted_

 154 extern const double _TBL_exp2_hi[], _TBL_exp2_lo[];
 155 static const double             /* poly app of 2^x-1 on [-1e-10,2^-7+1e-10] */
 156         E1 = 6.93147180559945310067495853381034619732 8e-0001,
 157         E2 = 2.40226506958777934784676915171749381597 9e-0001,
 158         E3 = 5.55041086647541051263112489277393786469 9e-0002,
 159         E4 = 9.61814320999102682485371274016245142335 5e-0003,
 160         E5 = 1.33335767654994034509677412223184908299 1e-0003;

 162 double
 163 pow(double x, double y) {
 164         double z, ax;
 165         double y1, y2, w1, w2;
 166         int sbx, sby, j, k, yisint;
 167         int hx, hy, ahx, ahy;
 168         unsigned lx, ly;
 169         int *pz = (int *) &z;

 171         hx = ((int *) &x)[HIWORD];
 172         lx = ((unsigned *) &x)[LOWORD];
 173         hy = ((int *) &y)[HIWORD];
 174         ly = ((unsigned *) &y)[LOWORD];
 175         ahx = hx & ~0x80000000;
 176         ahy = hy & ~0x80000000;
 177         if ((ahy | ly) == 0) {  /* y==zero  */
 178                 if ((ahx | lx) == 0)
 179                         z = _SVID_libm_err(x, y, 20);   /* +-0**+-0 */
 180                 else if ((ahx | (((lx | -lx) >> 31) & 1)) > 0x7ff00000)
 181                         z = _SVID_libm_err(x, y, 42);   /* NaN**+-0 */
 182                 else
 183                         z = one;                        /* x**+-0 = 1 */
 184                 return (z);
 185         } else if (hx == 0x3ff00000 && lx == 0 &&
 186                 (__xpg6 & _C99SUSv3_pow) != 0)
 187                 return (one);                   /* C99: 1**anything = 1 */
 188         else if (ahx > 0x7ff00000 || (ahx == 0x7ff00000 && lx != 0) ||
 189                 ahy > 0x7ff00000 || (ahy == 0x7ff00000 && ly != 0))
 190                 return (x * y); /* +-NaN return x*y; + -> * for Cheetah */
 191                                 /* includes Sun: 1**NaN = NaN */
 192         sbx = (unsigned) hx >> 31;
 193         sby = (unsigned) hy >> 31;
 194         ax = fabs(x);

 196         /*
 197          * determine if y is an odd int when x < 0
 198          * yisint = 0 ... y is not an integer
 199          * yisint = 1 ... y is an odd int
 200          * yisint = 2 ... y is an even int
 201          */
 202         yisint = 0;
 203         if (sbx) {
 204                 if (ahy >= 0x43400000)
 205                         yisint = 2;             /* even integer y */
 206                 else if (ahy >= 0x3ff00000) {
 207                         k = (ahy >> 20) - 0x3ff;        /* exponent */
 208                         if (k > 20) {
 209                                 j = ly >> (52 - k);
 210                                 if ((j << (52 - k)) == ly)
 211                                         yisint = 2 - (j & 1);
 212                         } else if (ly == 0) {
 213                                 j = ahy >> (20 - k);
```



```
 214                                 if ((j << (20 - k)) == ahy)
 215                                         yisint = 2 - (j & 1);
 216                         }
 217                 }
 218         }
 219         /* special value of y */
 220         if (ly == 0) {
 221                 if (ahy == 0x7ff00000) {        /* y is +-inf */
 222                         if (((ahx - 0x3ff00000) | lx) == 0) {
 223                                 if ((__xpg6 & _C99SUSv3_pow) != 0)
 224                                         return (one);
 225                                                 /* C99: (-1)**+-inf = 1 */
 226                                 else
 227                                         return (y - y);
 228                                                 /* Sun: (+-1)**+-inf = NaN */
 229                         } else if (ahx >= 0x3ff00000)
 230                                                 /* (|x|>1)**+,-inf = inf,0 */
 231                                 return (sby == 0 ? y : zero);
 232                         else                    /* (|x|<1)**-,+inf = inf,0 */
 233                                 return (sby != 0 ? -y : zero);
 234                 }
 235                 if (ahy == 0x3ff00000) {        /* y is  +-1 */
 236                         if (sby != 0) { /* y is -1 */
 237                                 if (x == zero)  /* divided by zero */
 238                                         return (_SVID_libm_err(x, y, 23));
 239                                 else if (ahx < 0x40000 || ((ahx - 0x40000) |
 240                                         lx) == 0)       /* overflow */
 241                                         return (_SVID_libm_err(x, y, 21));
 242                                 else
 243                                         return (one / x);
 244                         } else
 245                                 return (x);
 246                 }
 247                 if (hy == 0x40000000) {         /* y is  2 */
 248                         if (ahx >= 0x5ff00000 && ahx < 0x7ff00000)
 249                                 return (_SVID_libm_err(x, y, 21));
 250                                                         /* x*x overflow */
 251                         else if ((ahx < 0x1e56a09e && (ahx | lx) != 0) ||
 252                                 (ahx == 0x1e56a09e && lx < 0x667f3bcd))
 251                         else if (ahx < 0x1e56a09e && (ahx | lx) != 0 ||
 252                                 ahx == 0x1e56a09e && lx < 0x667f3bcd)
 253                                 return (_SVID_libm_err(x, y, 22));
 254                                                         /* x*x underflow */
 255                         else
 256                                 return (x * x);
 257                 }
 258                 if (hy == 0x3fe00000) {
 259                         if (!((ahx | lx) == 0 || ((ahx - 0x7ff00000) | lx) ==
 260                                 0 || sbx == 1))
 261                                 return (sqrt(x));       /* y is 0.5 and x > 0 */
 262                 }
 263         }
 264         /* special value of x */
 265         if (lx == 0) {
 266                 if (ahx == 0x7ff00000 || ahx == 0 || ahx == 0x3ff00000) {
 267                         /* x is +-0,+-inf,-1 */
 268                         z = ax;
 269                         if (sby == 1) {
 270                                 z = one / z;    /* z = |x|**y */
 271                                 if (ahx == 0)
 272                                         return (_SVID_libm_err(x, y, 23));
 273                         }
 274                         if (sbx == 1) {
 275                                 if (ahx == 0x3ff00000 && yisint == 0)
 276                                         z = _SVID_libm_err(x, y, 24);
 277                                         /* neg**non-integral is NaN + invalid */
```

```
278                                 else if (yisint == 1)
279                                         z = -z; /* (x<0)**odd = -(|x|**odd) */
280                         }
281                         return (z);
282                 }
283         }
284         /* (x<0)**(non-int) is NaN */
285         if (sbx == 1 && yisint == 0)
286                 return (_SVID_libm_err(x, y, 24));
287         /* Now ax is finite, y is finite */
288         /* first compute log2(ax) = w1+w2, with 24 bits w1 */
289         w1 = log2_x(ax, &w2);

291         /* split up y into y1+y2 and compute (y1+y2)*(w1+w2) */
292         if (((ly & 0x07ffffff) == 0) || ahy >= 0x47e00000 ||
293                 ahy <= 0x38100000) {
294                 /* no need to split if y is short or too large or too small */
295                 y1 = y * w1;
296                 y2 = y * w2;
297         } else {
298                 y1 = (double) ((float) y);
299                 y2 = (y - y1) * w1 + y * w2;
300                 y1 *= w1;
301         }
302         z = y1 + y2;
303         j = pz[HIWORD];
304         if (j >= 0x40900000) {                          /* z >= 1024 */
305                 if (!(j == 0x40900000 && pz[LOWORD] == 0))      /* z > 1024 */
306                         return (_SVID_libm_err(x, y, 21));      /* overflow */
307                 else {
308                         w2 = y1 - z;
309                         w2 += y2;
310                                                         /* rounded to inf */
311                         if (w2 >= -8.008566259537296567160e-17)
312                                 return (_SVID_libm_err(x, y, 21));
313                                                         /* overflow */
314                 }
315         } else if ((j & ~0x80000000) >= 0x4090cc00) {   /* z <= -1075 */
316                 if (!(j == 0xc090cc00 && pz[LOWORD] == 0))      /* z < -1075 */
317                         return (_SVID_libm_err(x, y, 22));      /* underflow */
318                 else {
319                         w2 = y1 - z;
320                         w2 += y2;
321                         if (w2 <= zero)                 /* underflow */
322                                 return (_SVID_libm_err(x, y, 22));
323                 }
324         }
325         /*
326          * compute 2**(k+f[j]+g)
327          */
328         k = (int) (z * 64.0 + (((hy ^ (ahx - 0x3ff00000)) > 0) ? 0.5 : -0.5));
329         j = k & 63;
330         w1 = y2 - ((double) k * 0.015625 - y1);
331         w2 = _TBL_exp2_hi[j];
332         z = _TBL_exp2_lo[j] + (w2 * w1) * (E1 + w1 * (E2 + w1 * (E3 + w1 *
333                 (E4 + w1 * E5))));
334         z += w2;
335         k >>= 6;
336         if (k < -1021)
337                 z = scalbn(z, k);
338         else                    /* subnormal output */
339                 pz[HIWORD] += k << 20;
340         if (sbx == 1 && yisint == 1)
341                 z = -z;         /* (-ve)**(odd int) */
342         return (z);
343 }
```

```
**********************************************************
    1935 Sun May 11 12:15:36 2014
new/usr/src/lib/libm/common/LD/__rem_pio2l.c
**********************************************************
```

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

/* __rem_pio2l(x,y)
 *
 * return the remainder of x rem pi/2 in y[0]+y[1]
 * by calling __rem_pio2m
 */

#include "libm.h"

#include "longdouble.h"

extern const int _TBL_ipio2l_inf[];

static const long double
    two24l = 16777216.0L,
    pio4   = 0.7853981633974483096156608458198757210495L;

int
__rem_pio2l(long double x, long double *y)
{
        long double     z, w;
        double          t[3], v[5];
        int             e0, i, nx, n, sign;

        sign = signbitl(x);
        z = fabsl(x);
        if (z <= pio4) {
                y[0] = x;
                y[1] = 0;
                return (0);
        }
        e0 = ilogbl(z) - 23;
        z = scalbnl(z, -e0);
        for (i = 0; i < 3; i++) {
```



```
                t[i] = (double)((int)(z));
                z = (z - (long double)t[i]) * two24l;
        }
        nx = 3;
        while (t[nx-1] == 0.0)
                nx--;   /* omit trailing zeros */
        n = __rem_pio2m(t, v, e0, nx, 2, _TBL_ipio2l_inf);
        z = (long double)v[1];
        w = (long double)v[0];
        y[0] = z + w;
        y[1] = z - (y[0] - w);
        if (sign == 1) {
                y[0] = -y[0];
                y[1] = -y[1];
                return (-n);
        }
        return (n);
}
```

*_____unchanged_portion_omitted_*

```
**********************************************************
    5472 Sun May 11 12:15:37 2014
new/usr/src/lib/libm/common/LD/__tanl.c
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" 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_tanl( long double x;  long double y; int k )
 * kernel tan/cotan function on [-pi/4, pi/4], pi/4 ~ 0.785398164
 * Input x is assumed to be bounded by ~pi/4 in magnitude.
 * Input y is the tail of x.
 * Input k indicate -- tan if k=0; else -1/tan
 *
 * Table look up algorithm
 *      1. by tan(-x) = -tan(x), need only to consider positive x
 *      2. if x < 5/32 = [0x3ffc4000, 0] = 0.15625 , then
 *           if x < 2^-57 (hx < 0x3fc40000 0), set w=x with inexact if x!= 0
 *           else
 *              z = x*x;
 *              w = x + (y+(x*z)*(t1+z*(t2+z*(t3+z*(t4+z*(t5+z*t6))))))
 *         return (k == 0 ? w : 1/w);
 *      3. else
 *              ht = (hx + 0x400)&0x7ffff800    (round x to a break point t)
 *              lt = 0
 *              i  = (hy-0x3ffc4000)>>11;       (i<=64)
 *              x' = (x - t)+y                  (|x'| ~<= 2^-7)
 *         By
 *              tan(t+x')
 *                = (tan(t)+tan(x'))/(1-tan(x')tan(t))
 *         We have
 *                              sin(x')+tan(t)*(tan(t)*sin(x'))
 *                = tan(t) + ------------------------------    for k=0
 *                                 cos(x') - tan(t)*sin(x')
 *
 *                             cos(x') - tan(t)*sin(x')
 *                = - ----------------------------------    for k=1
 *                      tan(t) + tan(t)*(cos(x')-1) + sin(x')
```



```
 *
 *         where        tan(t) is from the table,
 *                      sin(x') = x + pp1*x^3 + ...+ pp5*x^11
 *                      cos(x') = 1 + qq1*x^2 + ...+ qq5*x^10
 */

#include "libm.h"

#include <sys/isa_defs.h>

extern const long double _TBL_tanl_hi[], _TBL_tanl_lo[];
static const long double
one     = 1.0,
/*
 * |sin(x) - (x+pp1*x^3+...+ pp5*x^11)| <= 2^-122.32 for |x|<1/64
 */
pp1     = -1.66666666666666666666666666586782940810e-0001L,
pp2     =  8.33333333333333333333003723660929317540e-0003L,
pp3     = -1.98412698412698407604590348377833780447 0e-0004L,
pp4     =  2.75573192236190664131972310621090094941 3e-0006L,
pp5     = -2.50519839857094701909399846913501205767 3e-0008L,
/*
 *                      2           10        -123.84
 * |cos(x) - (1+qq1*x +...+ qq5*x  )| <= 2         for |x|<=1/128
 */
qq1     = -4.99999999999999999999999999999378373641e-0001L,
qq2     =  4.16666666666666666666665478399327703130e-0002L,
qq3     = -1.38888888888888888880582112306180516134 94e-0003L,
qq4     =  2.48015873015610537777158565890530311186 6e-0005L,
qq5     = -2.75572809976252632573648837669515700873 6e-0007L,
/*
 * |tan(x) - (x+t1*x^3+...+t6*x^13)|
 * |------------------------------ | <= 2^-59.73 for |x|<0.15625
 * |                x              |
 */
t1      =  3.33333333333333333333333333333423342490e-0001L,
t2      =  1.33333333333333333333333333309383874453 7e-0001L,
t3      =  5.39682539682539682539682790631868266225 0e-0002L,
t4      =  2.18694885361552028218557697699441848691 1e-0002L,
t5      =  8.86323552990219657335455451999115293624 6e-0003L,
t6      =  3.59212803657248006465219142754399487879 0e-0003L,
t7      =  1.45583438705145525785683380758190130547 4e-0003L,
t8      =  5.90027440931859985782998325620172558747 7e-0004L,
t9      =  2.39129115211726518150111696190112236293 7e-0004L,
t10     =  9.69153316938272974239402417319498188237 5e-0005L,
t11     =  3.92799473318641560322817818422578085995 1e-0005L,
t12     =  1.58830001884832382422764006488334101288 e-0005L,
t13     =  6.91627122339680831116620228513172223172 3e-0006L;
/* INDENT ON */
long double
__k_tanl(long double x, long double y, int k) {
        long double a, t, z, w = 0.0, s, c;
        long double a, t, z, w, s, c;
        int *pt = (int *) &t, *px = (int *) &x;
        int i, j, hx, ix;

        t = 1.0;
#if defined(__i386) || defined(__amd64)
        XTOI(px, hx);
#else
        hx = px[0];
#endif
        ix = hx & 0x7fffffff;
        if (ix < 0x3ffc4000) {
                if (ix < 0x3fc60000) {
                        if ((i = (int) x) == 0) /* generate inexact */
```

```
128                                     w = x;
129                 } else {
130                         z = x * x;
131                         if (ix < 0x3ff30000)    /* 2**-12 */
132                                 t = z * (t1 + z * (t2 + z * (t3 + z * t4)));
133                         else
134                                 t = z * (t1 + z * (t2 + z * (t3 + z * (t4 +
135                                         z * (t5 + z * (t6 + z * (t7 + z *
136                                         (t8 + z * (t9 + z * (t10 + z * (t11 +
137                                         z * (t12 + z * t13)))))))))));
138                         t = y + x * t;
139                         w = x + t;
140                 }
141                 return (k == 0 ? w : -one / w);
142         }
143         j = (ix + 0x400) & 0x7ffff800;
144         i = (j - 0x3ffc4000) >> 11;
145 #if defined(__i386) || defined(__amd64)
146         ITOX(j, pt);
147 #else
148         pt[0] = j;
149 #endif
150         if (hx > 0)
151                 x = y - (t - x);
152         else
153                 x = (-y) - (t + x);
154         a = _TBL_tanl_hi[i];
155         z = x * x;
156         /* cos(x)-1 */
157         t = z * (qq1 + z * (qq2 + z * (qq3 + z * (qq4 + z * qq5))));
158         /* sin(x) */
159         s = x * (one + z * (pp1 + z * (pp2 + z * (pp3 + z * (pp4 + z *
160                 pp5)))));
161         if (k == 0) {
162                 w = a * s;
163                 t = _TBL_tanl_lo[i] + (s + a * w) / (one - (w - t));
164                 return (hx < 0 ? -a - t : a + t);
165         } else {
166                 w = s + a * t;
167                 c = w + _TBL_tanl_lo[i];
168                 z = (one - (a * s - t));
169                 return (hx >= 0 ? z / (-a - c) : z / (a + c));
170         }
171 }
```

_____unchanged_portion_omitted_

```
**********************************************************
    1617 Sun May 11 12:15:39 2014
new/usr/src/lib/libm/common/LD/asinhl.c
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" 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;
        volatile long double dummy;
#endif /* ! codereview */

        w = fabsl(x);
        if (isnanl(x))
                return (x + x); /* x is NaN */
        if (w < tiny) {
#ifndef lint
                dummy = x + big;        /* inexact if x != 0 */
                volatile long double 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));
}
```

```
**********************************************************
    1612 Sun May 11 12:15:40 2014
new/usr/src/lib/libm/common/LD/isnanl.c
**********************************************************
______unchanged_portion_omitted_
  43 #elif defined(__x86)
  44 int
  45 isnanl(long double x) {
  46         int *px = (int *) &x, t = px[2] & 0x7fff;
  47 #if defined(HANDLE_UNSUPPORTED)
  48         return ((t == 0x7fff && ((px[1] & ~0x80000000) | px[0]) != 0) ||
  49                 (t != 0 && (px[1] & 0x80000000) == 0));
  48         return (t == 0x7fff && ((px[1] & ~0x80000000) | px[0]) != 0 ||
  49                 t != 0 && (px[1] & 0x80000000) == 0);
  50 #else
  51         return (t == 0x7fff && ((px[1] & ~0x80000000) | px[0]) != 0);
  52 #endif
  53 }
______unchanged_portion_omitted_
```

```
**********************************************************
    7031 Sun May 11 12:15:41 2014
new/usr/src/lib/libm/common/LD/jnl.c
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" 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;
        GENERIC a, b, temp, z, w;

    /* J(-n,x) = (-1)^n * J(n, x), J(n, -x) = (-1)^n * J(n, x)
     * Thus, J(-n,x) = J(n,-x)
     */
        if(n<0){
                n = -n;
                x = -x;
        }
        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;
                    case 2: temp = -cosl(x)-sinl(x); break;
                    case 3: temp =  cosl(x)-sinl(x); break;
                }
                b = invsqrtpi*temp/sqrtl(x);
            } else {
                a = j0l(x);
                b = j1l(x);
                for(i=1;i<n;i++){
                    temp = b;
                    b = b*((GENERIC)(i+i)/x) - a; /* avoid underflow */
                    a = temp;
                }
            }
        } else {
            if(x<1e-17L) {      /* use J(n,x) = 1/n!*(x/2)^n */
                b = powl(0.5L*x,(GENERIC) n);
```

```
128                 if (b!=zero) {
129                     for(a=one,i=1;i<=n;i++) a *= (GENERIC)i;
130                     b = b/a;
131                 }
132             } else {
133                 /* use backward recurrence */
134                 /*                         x      x^2      x^2
135                  *  J(n,x)/J(n-1,x) =  ----   ------   ------   .....
136                  *                         2n  - 2(n+1) - 2(n+2)
137                  *
138                  *                         1      1        1
139                  *  (for large x)   =  ----  ------   ------   .....
140                  *                         2n   2(n+1)   2(n+2)
141                  *                        -- - ------ - ------ -
142                  *                         x     x         x
143                  *
144                  * Let w = 2n/x and h=2/x, then the above quotient
145                  * is equal to the continued fraction:
146                  *                   1
147                  *      = -----------------------
148                  *                  1
149                  *         w - -----------------
150                  *                    1
151                  *              w+h - ---------
152                  *                    w+2h - ...
153                  *
154                  * To determine how many terms needed, let
155                  * Q(0) = w, Q(1) = w(w+h) - 1,
156                  * Q(k) = (w+k*h)*Q(k-1) - Q(k-2),
157                  * When Q(k) > 1e4      good for single
158                  * When Q(k) > 1e9      good for double
159                  * When Q(k) > 1e17     good for quaduple
160                  */
161             /* determin k */
162                 GENERIC t,v;
163                 double q0,q1,h,tmp; int k,m;
164                 w  = (n+n)/(double)x; h = 2.0/(double)x;
165                 q0 = w;  z = w+h; q1 = w*z - 1.0; k=1;
166                 while(q1<1.0e17) {
167                         k += 1; z += h;
168                         tmp = z*q1 - q0;
169                         q0 = q1;
170                         q1 = tmp;
171                 }
172                 m = n+n;
173                 for(t=zero, i = 2*(n+k); i>=m; i -= 2) t = one/(i/x-t);
174                 a = t;
175                 b = one;
176                 /* estimate log((2/x)^n*n!) = n*log(2/x)+n*ln(n)
177                    hence, if n*(log(2n/x)) > ...
178                     single 8.8722839355e+01
179                     double 7.09782712893383973096e+02
180                     long double 1.13565234062941439494919310779707650060170e+04
181                     then recurrent value may overflow and the result is
182                     likely underflow to zero
183                  */
184                 tmp = n;
185                 v = two/x;
186                 tmp = tmp*logl(fabsl(v*tmp));
187                 if(tmp<1.13565234062941439494919310779707650060L-04L) {
188                     for(i=n-1;i>0;i--){
189                         temp = b;
190                         b = ((i+i)/x)*b - a;
191                         a = temp;
192                     }
193                 } else {
```



```
194                     for(i=n-1;i>0;i--){
195                         temp = b;
196                         b = ((i+i)/x)*b - a;
197                         a = temp;
198                         if(b>1e1000L) {
199                             a /= b;
200                             t /= b;
201                             b  = 1.0;
202                         }
203                     }
204                 }
205                 b = (t*j0l(x)/b);
206             }
207         }
208         if(sgn==1) return -b; else return b;
209 }

211 GENERIC ynl(n,x)
212 int n; GENERIC x;{
213         int i;
214         int sign;
215         GENERIC a, b, temp = 0;
215         GENERIC a, b, temp;

217         if(x!=x)
218                 return x+x;
219         if (x <= zero) {
217         if(x!=x) return x+x;
218         if (x <= zero)
220                 if(x==zero)
221                         return -one/zero;
222                 else
223                         return zero/zero;
224         }
225 #endif /* ! codereview */
226         sign = 1;
227         if(n<0){
228                 n = -n;
229                 if((n&1) == 1) sign = -1;
230         }
231         if(n==0) return(y0l(x));
232         if(n==1) return(sign*y1l(x));
233         if(!finitel(x)) return zero;

235         if(x>1.0e91L) { /* x >> n**2
236                                     Jn(x) = cos(x-(2n+1)*pi/4)*sqrt(2/x*pi)
237                                     Yn(x) = sin(x-(2n+1)*pi/4)*sqrt(2/x*pi)
238                                     Let s=sin(x), c=cos(x),
239                                         xn=x-(2n+1)*pi/4, sqt2 = sqrt(2),then

241                                            n    sin(xn)*sqt2    cos(xn)*sqt2
242                                         ----------------------------------
243                                            0     s-c             c+s
244                                            1    -s-c            -c+s
245                                            2    -s+c            -c-s
246                                            3     s+c             c-s
247                                 */
248                 switch(n&3) {
249                     case 0: temp =  sinl(x)-cosl(x); break;
250                     case 1: temp = -sinl(x)-cosl(x); break;
251                     case 2: temp = -sinl(x)+cosl(x); break;
252                     case 3: temp =  sinl(x)+cosl(x); break;
253                 }
254                 b = invsqrtpi*temp/sqrtl(x);
255         } else {
256                 a = y0l(x);
```

```
257                 b = y1l(x);
258                 /*
259                  * fix 1262058 and take care of non-default rounding
260                  */
261                 for (i = 1; i < n; i++) {
262                         temp = b;
263                         b *= (GENERIC) (i + i) / x;
264                         if (b <= -LDBL_MAX)
265                                 break;
266                         b -= a;
267                         a = temp;
268                 }
269         }
270         if(sign>0) return b; else return -b;
271 }
```

```
**********************************************************
    1619 Sun May 11 12:15:42 2014
new/usr/src/lib/libm/common/LD/log1pl.c
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" 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 log1pl = __log1pl
#endif

/*
 * log1pl(x)
 * Kahan's trick based on log(1+x)/x being a slow varying function.
 */

#include "libm.h"

#if defined(__x86)
#define __swapRD        __swap87RD
#endif
extern enum fp_direction_type __swapRD(enum fp_direction_type);

long double
log1pl(long double x) {
        long double y;
        enum fp_direction_type rd;

        if (x != x)
                return (x + x);
        if (x < -1.L)
                return (logl(x));
        rd = __swapRD(fp_nearest);
        y = 1.L + x;
        if (y != 1.L) {
        if (y != 1.L)
                if (y == x)
                        x = logl(x);
                else
                        x *= logl(y) / (y - 1.L);
```



```
        }
#endif /* ! codereview */
        if (rd != fp_nearest)
                (void) __swapRD(rd);
        return (x);
}
```

```
**********************************************************
    1768 Sun May 11 12:15:44 2014
new/usr/src/lib/libm/common/LD/scalbl.c
**********************************************************
```

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

/*
 * scalbl(x,n): return x * 2**n by manipulating exponent.
 */

#include "libm.h"
#include "longdouble.h"

#include <sys/isa_defs.h>

long double
scalbl(long double x, long double fn) {
        int *py = (int *) &fn, n;
        long double z;

        if (isnanl(x) || isnanl(fn))
                return x * fn;

        /* fn is +/-Inf */
#if defined(_BIG_ENDIAN)
        if ((py[0] & 0x7fff0000) == 0x7fff0000) {
        if ((py[0] & 0x7fff0000) == 0x7fff0000)
                if ((py[0] & 0x80000000) != 0)
#else
        if ((py[2] & 0x7fff) == 0x7fff) {
        if ((py[2] & 0x7fff) == 0x7fff)
                if ((py[2] & 0x8000) != 0)
#endif
                        return x / (-fn);
                else
                        return x * fn;
        }
```



```
        if (rintl(fn) != fn)
                return (fn - fn) / (fn - fn);
        if (fn > 65000.0L)
                z = scalbnl(x, 65000);
        else if (-fn > 65000.0L)
                z = scalbnl(x, -65000);
        else {
                n = (int) fn;
                z = scalbnl(x, n);
        }
        return z;
}
```

_____***unchanged_portion_omitted_***

****
2608 Sun May 11 12:15:46 2014
new/usr/src/lib/libm/common/LD/tanhl.c
****

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

        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;
                        volatile long double 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);
}
```

_____***unchanged_portion_omitted_***

```
**********************************************************
    5459 Sun May 11 12:15:48 2014
new/usr/src/lib/libm/common/Q/__tanl.c
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" 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_tanl(long double x; long double y, int k);
 * kernel tan/cotan function on [-pi/4, pi/4], pi/4 ~ 0.785398164
 * Input x is assumed to be bounded by ~pi/4 in magnitude.
 * Input y is the tail of x.
 * Input k indicate -- tan if k=0; else -1/tan
 *
 * Table look up algorithm
 *      1. by tan(-x) = -tan(x), need only to consider positive x
 *      2. if x < 5/32 = [0x3ffc4000, 0] = 0.15625 , then
 *           if x < 2^-57 (hx < 0x3fc40000 0), set w=x with inexact if x!= 0
 *           else
 *              z = x*x;
 *              w = x + (y+(x*z)*(t1+z*(t2+z*(t3+z*(t4+z*(t5+z*t6))))))
 *         return (k==0)? w: 1/w;
 *      3. else
 *              ht = (hx + 0x400)&0x7ffff800    (round x to a break point t)
 *              lt = 0
 *              i  = (hy-0x3ffc4000)>>11;       (i<=64)
 *              x' = (x - t)+y                  (|x'| ~<= 2^-7)
 *         By
 *              tan(t+x')
 *                = (tan(t)+tan(x'))/(1-tan(x')tan(t))
 *         We have
 *                                sin(x')+tan(t)*(tan(t)*sin(x'))
 *                = tan(t) + ------------------------------    for k=0
 *                              cos(x') - tan(t)*sin(x')
 *
 *                              cos(x') - tan(t)*sin(x')
 *                = - ----------------------------------    for k=1
 *                     tan(t) + tan(t)*(cos(x')-1) + sin(x')
 *
 *
```



```
 *         where        tan(t) is from the table,
 *                      sin(x') = x + pp1*x^3 + ...+ pp5*x^11
 *                      cos(x') = 1 + qq1*x^2 + ...+ qq5*x^10
 */

#include "libm.h"

extern const long double _TBL_tanl_hi[], _TBL_tanl_lo[];
static const long double
        one     = 1.0L,
/*
 *                      3           11       -122.32
 * |sin(x) - (x+pp1*x +...+ pp5*x  )| <= 2        for |x|<1/64
 */
        pp1     = -1.666666666666666666666666666586782940810e-0001L,
        pp2     = +8.333333333333333333333003723660929317540e-0003L,
        pp3     = -1.984126984126984076045903483778337804470e-0004L,
        pp4     = +2.755731922361906641319723106210900949413e-0006L,
        pp5     = -2.505198398570947019093998469135012057673e-0008L,
/*
 *                      2           10        -123.84
 * |cos(x) - (1+qq1*x +...+ qq5*x  )| <= 2        for |x|<=1/128
 */
        qq1     = -4.999999999999999999999999999999378373641e-0001L,
        qq2     = +4.166666666666666666666665478399327703130e-0002L,
        qq3     = -1.388888888888888880582112306180516134940e-0003L,
        qq4     = +2.480158730156105377771585658905303111866e-0005L,
        qq5     = -2.755728099762526325736488376695157008736e-0007L,
/*
 * |tan(x) - (x+t1*x^3+...+t6*x^13)|
 * |------------------------------ | <= 2^-59.73 for |x|<0.15625
 * |               x               |
 */
        t1      = +3.333333333333333333333333333423342490e-0001L,
        t2      = +1.333333333333333333333333333093838744537e-0001L,
        t3      = +5.396825396825396825396827906318682662250e-0002L,
        t4      = +2.186948853615520282185576976994418486911e-0002L,
        t5      = +8.863235529902196573354554519991152936246e-0003L,
        t6      = +3.592128036572480064652191427543994878790e-0003L,
        t7      = +1.455834387051455257856833807581901305474e-0003L,
        t8      = +5.900274409318599857829983256201725587477e-0004L,
        t9      = +2.391291152117265181501116961901122362937e-0004L,
        t10     = +9.691533169382729742394024173194981882375e-0005L,
        t11     = +3.927994733186415603228178184225780859951e-0005L,
        t12     = +1.588300018848323824227640064883334101288e-0005L,
        t13     = +6.916271223396808311166202285131722231723e-0006L;

#define i0      0

long double
__k_tanl(long double x, long double y, int k) {
        long double a, t, z, w = 0, s, c;
        long double a, t, z, w, s, c;
        int *pt = (int *) &t, *px = (int *) &x;
        int i, j, hx, ix;

        t = 1.0L;
        hx = px[i0];
        ix = hx & 0x7fffffff;
        if (ix < 0x3ffc4000) {
                *(3 - i0 + (int *) &t) = 1;     /* make t = one+ulp */
                if (ix < 0x3fc60000) {
                        if (((int) (x * t)) < 1)        /* generate inexact */
                                w = x;  /* generate underflow if subnormal */
                } else {
                        z = x * x;
```

```
128                         if (ix < 0x3ff30000)    /* 2**-12 */
129                                 t = z * (t1 + z * (t2 + z * (t3 + z * t4)));
130                         else
131                                 t = z * (t1 + z * (t2 + z * (t3 + z * (t4 +
132                                         z * (t5 + z * (t6 + z * (t7 + z * (t8 +
133                                         z * (t9 + z * (t10 + z * (t11 +
134                                         z * (t12 + z * t13)))))))))));
135                         t = y + x * t;
136                         w = x + t;
137                 }
138                 return (k == 0 ? w : -one / w);
139         }
140         j = (ix + 0x400) & 0x7ffff800;
141         i = (j - 0x3ffc4000) >> 11;
142         pt[i0] = j;
143         if (hx > 0)
144                 x = y - (t - x);
145         else
146                 x = (-y) - (t + x);
147         a = _TBL_tanl_hi[i];
148         z = x * x;
149         /* cos(x)-1 */
150         t = z * (qq1 + z * (qq2 + z * (qq3 + z * (qq4 + z * qq5))));
151         /* sin(x) */
152         s = x * (one + z * (pp1 + z * (pp2 + z * (pp3 + z * (pp4 + z * pp5)))));
153         if (k == 0) {
154                 w = a * s;
155                 t = _TBL_tanl_lo[i] + (s + a * w) / (one - (w - t));
156                 return (hx < 0 ? -a - t : a + t);
157         } else {
158                 w = s + a * t;
159                 c = w + _TBL_tanl_lo[i];
160                 z = one - (a * s - t);
161                 return (hx >= 0 ? z / (-a - c) : z / (a + c));
162         }
163 }
```

*_____unchanged_portion_omitted_*

```
**********************************************************
    1617 Sun May 11 12:15:49 2014
new/usr/src/lib/libm/common/Q/asinhl.c
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" 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;
        volatile long double dummy;
#endif /* ! codereview */

        w = fabsl(x);
        if (isnanl(x))
                return (x + x); /* x is NaN */
        if (w < tiny) {
#ifndef lint
                dummy = x + big;        /* inexact if x != 0 */
                volatile long double 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));
}
```

****
2037 Sun May 11 12:15:50 2014
new/usr/src/lib/libm/common/Q/asinl.c
****

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

/*
 *      asinl(x) = atan2l(x,sqrt(1-x*x));
 *
 * For better accuracy, 1-x*x is computed as follows
 *      1-x*x                   if x <  0.5,
 *      2*(1-|x|)-(1-|x|)*(1-|x|) if x >= 0.5.
 *
 * Special cases:
 *      if x is NaN, return x itself;
 *      if |x|>1, return NaN with invalid signal.
 */

#include "libm.h"

static const long double zero = 0.0L, small = 1.0e-20L, half = 0.5L, one = 1.0L;
#ifndef lint
static const long double big = 1.0e+20L;
#endif

long double
asinl(long double x) {
        long double t, w;
        volatile long double dummy;
#endif /* ! codereview */

        w = fabsl(x);
        if (isnanl(x))
                return (x + x);
        else if (w <= half) {
```



```
                if (w < small) {
#ifndef lint
                        dummy = w + big;
                        volatile long double dummy = w + big;
                                                        /* inexact if w != 0 */
#endif
                        return (x);
                } else
                        return (atanl(x / sqrtl(one - x * x)));
        } else if (w < one) {
                t = one - w;
                w = t + t;
                return (atanl(x / sqrtl(w - t * t)));
        } else if (w == one)
                return (atan2l(x, zero));       /* asin(+-1) =  +- PI/2 */
        else
                return (zero / zero);           /* |x| > 1: invalid */
}
```

unchanged_portion_omitted_

```
**********************************************************
    4154 Sun May 11 12:15:52 2014
new/usr/src/lib/libm/common/Q/atan2l.c
**********************************************************
```

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

/*
 * atan2l(y,x)
 *
 * Method :
 *      1. Reduce y to positive by atan2(y,x)=-atan2(-y,x).
 *      2. Reduce x to positive by (if x and y are unexceptional):
 *              ARG (x+iy) = arctan(y/x)           ... if x > 0,
 *              ARG (x+iy) = pi - arctan[y/(-x)]   ... if x < 0,
 *
 * Special cases:
 *
 *      ATAN2((anything), NaN ) is NaN;
 *      ATAN2(NAN , (anything) ) is NaN;
 *      ATAN2(+-0, +(anything but NaN)) is +-0  ;
 *      ATAN2(+-0, -(anything but NaN)) is +-PI ;
 *      ATAN2(+-(anything but 0 and NaN), 0) is +-PI/2;
 *      ATAN2(+-(anything but INF and NaN), +INF) is +-0 ;
 *      ATAN2(+-(anything but INF and NaN), -INF) is +-PI;
 *      ATAN2(+-INF,+INF ) is +-PI/4 ;
 *      ATAN2(+-INF,-INF ) is +-3PI/4;
 *      ATAN2(+-INF, (anything but,0,NaN, and INF)) is +-PI/2;
 *
 * Constants:
 * The hexadecimal values are the intended ones for the following constants.
 * The decimal values may be used, provided that the compiler will convert
 * from decimal to binary accurately enough to produce the hexadecimal values
 * shown.
 */

#pragma weak atan2l = __atan2l

#include "libm.h"
#include "longdouble.h"
```



```
static const long double
        zero    =  0.0L,
        tiny    =  1.0e-40L,
        one     =  1.0L,
        half    =  0.5L,
        PI3o4   =  2.3561944901923449288469825374596271631 48L,
        PIo4    =  0.7853981633974483096156608458198757210 49L,
        PIo2    =  1.5707963267948966192313216916397514420 99L,
        PI      =  3.1415926535897932384626433832795028841 97L,
        PI_lo   =  8.6718101301237810247970440260433519687 62e-35L;

long double
atan2l(long double y, long double x) {
        long double t, z;
        int k, m, signy, signx;

        if (x != x || y != y)
                return (x + y); /* return NaN if x or y is NAN */
        signy = signbitl(y);
        signx = signbitl(x);
        if (x == one)
                return (atanl(y));
        m = signy + signx + signx;

        /* when y = 0 */
        if (y == zero)
                switch (m) {
                case 0:
                        return (y);     /* atan(+0,+anything) */
                case 1:
                        return (y);     /* atan(-0,+anything) */
                case 2:
                        return (PI + tiny);     /* atan(+0,-anything) */
                case 3:
                        return (-PI - tiny);    /* atan(-0,-anything) */
                }

        /* when x = 0 */
        if (x == zero)
                return (signy == 1 ? -PIo2 - tiny : PIo2 + tiny);

        /* when x is INF */
        if (!finitel(x)) {
        if (!finitel(x))
                if (!finitel(y)) {
                        switch (m) {
                        case 0:
                                return (PIo4 + tiny);   /* atan(+INF,+INF) */
                        case 1:
                                return (-PIo4 - tiny);  /* atan(-INF,+INF) */
                        case 2:
                                return (PI3o4 + tiny);  /* atan(+INF,-INF) */
                        case 3:
                                return (-PI3o4 - tiny); /* atan(-INF,-INF) */
                        }
                } else {
                        switch (m) {
                        case 0:
                                return (zero);  /* atan(+...,+INF) */
                        case 1:
                                return (-zero); /* atan(-...,+INF) */
                        case 2:
                                return (PI + tiny);     /* atan(+...,-INF) */
                        case 3:
                                return (-PI - tiny);    /* atan(-...,-INF) */
```

```
128                         }
129                 }
130         }

131         /* when y is INF */
132         if (!finitel(y))
133                 return (signy == 1 ? -PIo2 - tiny : PIo2 + tiny);

135         /* compute y/x */
136         x = fabsl(x);
137         y = fabsl(y);
138         t = PI_lo;
139         k = (ilogbl(y) - ilogbl(x));

141         if (k > 120)
142                 z = PIo2 + half * t;
143         else if (m > 1 && k < -120)
144                 z = zero;
145         else
146                 z = atanl(y / x);

148         switch (m) {
149         case 0:
150                 return (z);     /* atan(+,+) */
151         case 1:
152                 return (-z);    /* atan(-,+) */
153         case 2:
154                 return (PI - (z - t));  /* atan(+,-) */
155         case 3:
156                 return ((z - t) - PI);  /* atan(-,-) */
157         }
158         /* NOTREACHED */
159     return 0.0L;
160 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
    6994 Sun May 11 12:15:54 2014
new/usr/src/lib/libm/common/Q/jnl.c
**********************************************************
______unchanged_portion_omitted_

 214 GENERIC ynl(n,x)
 215 int n; GENERIC x;{
 216         int i;
 217         int sign;
 218         GENERIC a, b, temp;

 220         if(x!=x) return x+x;
 221         if (x <= zero) {
 221         if (x <= zero)
 222                 if(x==zero)
 223                         return -one/zero;
 224                 else
 225                         return zero/zero;
 226         }
 227 #endif /* ! codereview */
 228         sign = 1;
 229         if(n<0){
 230                 n = -n;
 231                 if((n&1) == 1) sign = -1;
 232         }
 233         if(n==0) return(y0l(x));
 234         if(n==1) return(sign*y1l(x));
 235         if(!finitel(x)) return zero;

 237         if(x>1.0e91L) { /* x >> n**2
 238                                 Jn(x) = cos(x-(2n+1)*pi/4)*sqrt(2/x*pi)
 239                                 Yn(x) = sin(x-(2n+1)*pi/4)*sqrt(2/x*pi)
 240                                 Let s=sin(x), c=cos(x),
 241                                     xn=x-(2n+1)*pi/4, sqt2 = sqrt(2),then

 243                                        n    sin(xn)*sqt2    cos(xn)*sqt2
 244                                     ----------------------------------
 245                                        0     s-c             c+s
 246                                        1    -s-c            -c+s
 247                                        2    -s+c            -c-s
 248                                        3     s+c             c-s
 249                                 */
 250                 switch(n&3) {
 251                     case 0: temp =  sinl(x)-cosl(x); break;
 252                     case 1: temp = -sinl(x)-cosl(x); break;
 253                     case 2: temp = -sinl(x)+cosl(x); break;
 254                     case 3: temp =  sinl(x)+cosl(x); break;
 255                 }
 256                 b = invsqrtpi*temp/sqrtl(x);
 257         } else {
 258                 a = y0l(x);
 259                 b = y1l(x);
 260                 /*
 261                  * fix 1262058 and take care of non-default rounding
 262                  */
 263                 for (i = 1; i < n; i++) {
 264                         temp = b;
 265                         b *= (GENERIC) (i + i) / x;
 266                         if (b <= -LDBL_MAX)
 267                                 break;
 268                         b -= a;
 269                         a = temp;
 270                 }
 271         }
 272         if(sign>0) return b; else return -b;
```



```
 273 }
```

```
**********************************************************
    2608 Sun May 11 12:15:55 2014
new/usr/src/lib/libm/common/Q/tanhl.c
**********************************************************
```

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

        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;
                        volatile long double 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);
}
```

_____*unchanged_portion_omitted_*

```
**********************************************************
    3022 Sun May 11 12:15:56 2014
new/usr/src/lib/libm/common/R/__tanf.c
**********************************************************
______unchanged_portion_omitted_
  58 /* INDENT ON */

  60 #define one q[0]
  61 #define P0 q[1]
  62 #define P1 q[2]
  63 #define P2 q[3]
  64 #define P3 q[4]
  65 #define P4 q[5]
  66 #define P5 q[6]
  67 #define P6 q[7]
  68 #define P7 q[8]
  69 #define T0 q[9]
  70 #define T1 q[10]

  72 float
  73 __k_tanf(double x, int n) {
  74         float ft = 0.0;
  74         float ft;
  75         double z, w;
  76         int ix;

  78         ix = ((int *) &x)[HIWORD] & ~0x80000000;        /* ix = leading |x| */
  79         /* small argument */
  80         if (ix < 0x3f800000) {          /* if |x| < 0.0078125 = 2**-7 */
  81                 if (ix < 0x3f100000) {  /* if |x| < 2**-14 */
  82                         if ((int) x == 0) {     /* raise inexact if x!=0 */
  83                                 ft = n == 0 ? (float) x : (float) (-one / x);
  84                         }
  85                         return (ft);
  86                 }
  87                 z = (x * T0) * (T1 + x * x);
  88                 ft = n == 0 ? (float) z : (float) (-one / z);
  89                 return (ft);
  90         }
  91         z = x * x;
  92         w = ((P0 * x) * (P1 + z * (P2 + z)) * (P3 + z * (P4 + z)))
  93                 * (P5 + z * (P6 + z * (P7 + z)));
  94         ft = n == 0 ? (float) w : (float) (-one / w);
  95         return (ft);
  96 }
______unchanged_portion_omitted_
```

```
**********************************************************
    3872 Sun May 11 12:15:57 2014
new/usr/src/lib/libm/common/R/cosf.c
**********************************************************
______unchanged_portion_omitted_

 59 #define S0      C[0]
 60 #define S1      C[1]
 61 #define S2      C[2]
 62 #define S3      C[3]
 63 #define C0      C[4]
 64 #define C1      C[5]
 65 #define C2      C[6]
 66 #define C3      C[7]
 67 #define C4      C[8]
 68 #define invpio2 C[9]
 69 #define half    C[10]
 70 #define pio2_1  C[11]
 71 #define pio2_t  C[12]

 73 float
 74 cosf(float x)
 75 {
 76         double  y, z, w;
 77         float   f;
 78         int     n, ix, hx, hy;
 79         volatile int i;
 80 #endif /* ! codereview */

 82         hx = *((int *)&x);
 83         ix = hx & 0x7fffffff;

 85         y = (double)x;

 87         if (ix <= 0x4016cbe4) {         /* |x| < 3*pi/4 */
 88                 if (ix <= 0x3f490fdb) {         /* |x| < pi/4 */
 89                         if (ix <= 0x39800000) { /* |x| <= 2**-12 */
 90                                 i = (int)y;
 79                                 volatile int    i = (int)y;
 91 #ifdef lint
 92                                 i = i;
 93 #endif
 94                                 return (1.0f);
 95                         }
 96                         z = y * y;
 97                         return ((float)(((C0 + z * C1) + (z * z) * C2) *
 98                             (C3 + z * (C4 + z))));
 99                 } else if (hx > 0) {
100                         y = (y - pio2_1) - pio2_t;
101                         z = y * y;
102                         return ((float)-((y * (S0 + z * S1)) *
103                             (S2 + z * (S3 + z))));
104                 } else {
105                         y = (y + pio2_1) + pio2_t;
106                         z = y * y;
107                         return ((float)((y * (S0 + z * S1)) *
108                             (S2 + z * (S3 + z))));
109                 }
110         } else if (ix <= 0x49c90fdb) {  /* |x| < 2^19*pi */
111 #if defined(__i386) && !defined(__amd64)
112                 int     rp;

114                 rp = __swapRP(fp_extended);
115 #endif
116                 w = y * invpio2;
117                 if (hx < 0)
```



```
118                         n = (int)(w - half);
119                 else
120                         n = (int)(w + half);
121                 y = (y - n * pio2_1) - n * pio2_t;
122                 n++;
123 #if defined(__i386) && !defined(__amd64)
124                 if (rp != fp_extended)
125                         (void) __swapRP(rp);
126 #endif
127         } else {
128                 if (ix >= 0x7f800000)
129                         return (x / x); /* cos(Inf or NaN) is NaN */
130                 hy = ((int *)&y)[HIWORD];
131                 n = ((hy >> 20) & 0x7ff) - 1046;
132                 ((int *)&w)[HIWORD] = (hy & 0xfffff) | 0x41600000;
133                 ((int *)&w)[LOWORD] = ((int *)&y)[LOWORD];
134                 n = __rem_pio2m(&w, &y, n, 1, 0, _TBL_ipio2_inf) + 1;
135         }

137         if (n & 1) {
138                 /* compute cos y */
139                 z = y * y;
140                 f = (float)(((C0 + z * C1) + (z * z) * C2) *
141                     (C3 + z * (C4 + z)));
142         } else {
143                 /* compute sin y */
144                 z = y * y;
145                 f = (float)((y * (S0 + z * S1)) * (S2 + z * (S3 + z)));
146         }

148         return ((n & 2)? -f : f);
149 }
______unchanged_portion_omitted_
```

```
**********************************************************
    5085 Sun May 11 12:15:58 2014
new/usr/src/lib/libm/common/R/sincosf.c
**********************************************************
______unchanged_portion_omitted_

  81 #define S0      C[0]
  82 #define S1      C[1]
  83 #define S2      C[2]
  84 #define S3      C[3]
  85 #define C0      C[4]
  86 #define C1      C[5]
  87 #define C2      C[6]
  88 #define C3      C[7]
  89 #define C4      C[8]
  90 #define invpio2 C[9]
  91 #define half    C[10]
  92 #define pio2_1  C[11]
  93 #define pio2_t  C[12]

  95 void
  96 sincosf(float x, float *s, float *c)
  97 {
  98         double  y, z, w;
  99         float   f, g;
 100         int     n, ix, hx, hy;
 101         volatile int i;
 102 #endif /* ! codereview */

 104         hx = *((int *)&x);
 105         ix = hx & 0x7fffffff;

 107         y = (double)x;

 109         if (ix <= 0x4016cbe4) {         /* |x| < 3*pi/4 */
 110                 if (ix <= 0x3f490fdb) {         /* |x| < pi/4 */
 111                         if (ix <= 0x39800000) { /* |x| <= 2**-12 */
 112                                 i = (int)y;
 101                                 volatile int    i = (int)y;
 113 #ifdef lint
 114                                 i = i;
 115 #endif
 116                                 *s = x;
 117                                 *c = 1.0f;
 118                                 return;
 119                         }
 120                         z = y * y;
 121                         *s = (float)((y * (S0 + z * S1)) *
 122                             (S2 + z * (S3 + z)));
 123                         *c = (float)(((C0 + z * C1) + (z * z) * C2) *
 124                             (C3 + z * (C4 + z)));
 125                 } else if (hx > 0) {
 126                         y = (y - pio2_1) - pio2_t;
 127                         z = y * y;
 128                         *s = (float)(((C0 + z * C1) + (z * z) * C2) *
 129                             (C3 + z * (C4 + z)));
 130                         *c = (float)-((y * (S0 + z * S1)) *
 131                             (S2 + z * (S3 + z)));
 132                 } else {
 133                         y = (y + pio2_1) + pio2_t;
 134                         z = y * y;
 135                         *s = (float)-(((C0 + z * C1) + (z * z) * C2) *
 136                             (C3 + z * (C4 + z)));
 137                         *c = (float)((y * (S0 + z * S1)) *
 138                             (S2 + z * (S3 + z)));
 139                 }
```



```
 140                 return;
 141         } else if (ix <= 0x49c90fdb) {  /* |x| < 2^19*pi */
 142 #if defined(__i386) && !defined(__amd64)
 143                 int     rp;

 145                 rp = __swapRP(fp_extended);
 146 #endif
 147                 w = y * invpio2;
 148                 if (hx < 0)
 149                         n = (int)(w - half);
 150                 else
 151                         n = (int)(w + half);
 152                 y = (y - n * pio2_1) - n * pio2_t;
 153 #if defined(__i386) && !defined(__amd64)
 154                 if (rp != fp_extended)
 155                         (void) __swapRP(rp);
 156 #endif
 157         } else {
 158                 if (ix >= 0x7f800000) {
 159                         *s = *c = x / x;
 160                         return;
 161                 }
 162                 hy = ((int *)&y)[HIWORD];
 163                 n = ((hy >> 20) & 0x7ff) - 1046;
 164                 ((int *)&w)[HIWORD] = (hy & 0xfffff) | 0x41600000;
 165                 ((int *)&w)[LOWORD] = ((int *)&y)[LOWORD];
 166                 n = __rem_pio2m(&w, &y, n, 1, 0, _TBL_ipio2_inf);
 167                 if (hy < 0) {
 168                         y = -y;
 169                         n = -n;
 170                 }
 171         }

 173         z = y * y;
 174         f = (float)((y * (S0 + z * S1)) * (S2 + z * (S3 + z)));
 175         g = (float)(((C0 + z * C1) + (z * z) * C2) *
 176             (C3 + z * (C4 + z)));
 177         if (n & 2) {
 178                 f = -f;
 179                 g = -g;
 180         }
 181         if (n & 1) {
 182                 *s = g;
 183                 *c = -f;
 184         } else {
 185                 *s = f;
 186                 *c = g;
 187         }
 188 }
______unchanged_portion_omitted_
```

```
**********************************************************
    3911 Sun May 11 12:16:00 2014
new/usr/src/lib/libm/common/R/sinf.c
**********************************************************
______unchanged_portion_omitted_

#define S0      C[0]
#define S1      C[1]
#define S2      C[2]
#define S3      C[3]
#define C0      C[4]
#define C1      C[5]
#define C2      C[6]
#define C3      C[7]
#define C4      C[8]
#define invpio2 C[9]
#define half    C[10]
#define pio2_1  C[11]
#define pio2_t  C[12]

float
sinf(float x)
{
        double  y, z, w;
        float   f;
        int     n, ix, hx, hy;
        volatile int i;
#endif /* ! codereview */

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

        y = (double)x;

        if (ix <= 0x4016cbe4) {         /* |x| < 3*pi/4 */
                if (ix <= 0x3f490fdb) {         /* |x| < pi/4 */
                        if (ix <= 0x39800000) { /* |x| <= 2**-12 */
                                i = (int)y;
                                volatile int    i = (int)y;
#ifdef lint
                                i = i;
#endif
                                return (x);
                        }
                        z = y * y;
                        return ((float)((y * (S0 + z * S1)) *
                            (S2 + z * (S3 + z))));
                } else if (hx > 0) {
                        y = (y - pio2_1) - pio2_t;
                        z = y * y;
                        return ((float)(((C0 + z * C1) + (z * z) * C2) *
                            (C3 + z * (C4 + z))));
                } else {
                        y = (y + pio2_1) + pio2_t;
                        z = y * y;
                        return ((float)-(((C0 + z * C1) + (z * z) * C2) *
                            (C3 + z * (C4 + z))));
                }
        } else if (ix <= 0x49c90fdb) {  /* |x| < 2^19*pi */
#if defined(__i386) && !defined(__amd64)
                int     rp;

                rp = __swapRP(fp_extended);
#endif
                w = y * invpio2;
                if (hx < 0)
```



```
                        n = (int)(w - half);
                else
                        n = (int)(w + half);
                y = (y - n * pio2_1) - n * pio2_t;
#if defined(__i386) && !defined(__amd64)
                if (rp != fp_extended)
                        (void) __swapRP(rp);
#endif
        } else {
                if (ix >= 0x7f800000)
                        return (x / x); /* sin(Inf or NaN) is NaN */
                hy = ((int *)&y)[HIWORD];
                n = ((hy >> 20) & 0x7ff) - 1046;
                ((int *)&w)[HIWORD] = (hy & 0xfffff) | 0x41600000;
                ((int *)&w)[LOWORD] = ((int *)&y)[LOWORD];
                n = __rem_pio2m(&w, &y, n, 1, 0, _TBL_ipio2_inf);
                if (hy < 0) {
                        y = -y;
                        n = -n;
                }
        }

        if (n & 1) {
                /* compute cos y */
                z = y * y;
                f = (float)(((C0 + z * C1) + (z * z) * C2) *
                    (C3 + z * (C4 + z)));
        } else {
                /* compute sin y */
                z = y * y;
                f = (float)((y * (S0 + z * S1)) * (S2 + z * (S3 + z)));
        }

        return ((n & 2)? -f : f);
}
______unchanged_portion_omitted_
```

```
**********************************************************
    4309 Sun May 11 12:16:01 2014
new/usr/src/lib/libm/common/R/tanf.c
**********************************************************
______unchanged_portion_omitted_

  57 #define one     C[0]
  58 #define P0      C[1]
  59 #define P1      C[2]
  60 #define P2      C[3]
  61 #define P3      C[4]
  62 #define P4      C[5]
  63 #define P5      C[6]
  64 #define P6      C[7]
  65 #define P7      C[8]
  66 #define T0      C[9]
  67 #define T1      C[10]
  68 #define invpio2 C[11]
  69 #define half    C[12]
  70 #define pio2_1  C[13]
  71 #define pio2_t  C[14]

  73 float
  74 tanf(float x)
  75 {
  76         double  y, z, w;
  77         float   f;
  78         int     n, ix, hx, hy;
  79         volatile int i;
  80 #endif /* ! codereview */

  82         hx = *((int *)&x);
  83         ix = hx & 0x7fffffff;

  85         y = (double)x;

  87         if (ix <= 0x4016cbe4) {         /* |x| < 3*pi/4 */
  88                 if (ix <= 0x3f490fdb) {         /* |x| < pi/4 */
  89                         if (ix < 0x3c000000) {          /* |x| < 2**-7 */
  90                                 if (ix <= 0x39800000) { /* |x| < 2**-12 */
  91                                         i = (int)y;
  79                                         volatile int    i = (int)y;
  92 #ifdef lint
  93                                         i = i;
  94 #endif
  95                                         return (x);
  96                                 }
  97                                 return ((float)((y * T0) * (T1 + y * y)));
  98                         }
  99                         z = y * y;
 100                         return ((float)(((P0 * y) * (P1 + z * (P2 + z)) *
 101                             (P3 + z * (P4 + z))) *
 102                             (P5 + z * (P6 + z * (P7 + z)))));
 103                 }
 104                 if (hx > 0)
 105                         y = (y - pio2_1) - pio2_t;
 106                 else
 107                         y = (y + pio2_1) + pio2_t;
 108                 hy = ((int *)&y)[HIWORD] & ~0x80000000;
 109                 if (hy < 0x3f800000) {          /* |y| < 2**-7 */
 110                         z = (y * T0) * (T1 + y * y);
 111                         return ((float)(-one / z));
 112                 }
 113                 z = y * y;
 114                 w = ((P0 * y) * (P1 + z * (P2 + z)) * (P3 + z * (P4 + z))) *
 115                     (P5 + z * (P6 + z * (P7 + z)));
```



```
 116                 return ((float)(-one / w));
 117         }

 119         if (ix <= 0x49c90fdb) { /* |x| < 2^19*pi */
 120 #if defined(__i386) && !defined(__amd64)
 121                 int     rp;

 123                 rp = __swapRP(fp_extended);
 124 #endif
 125                 w = y * invpio2;
 126                 if (hx < 0)
 127                         n = (int)(w - half);
 128                 else
 129                         n = (int)(w + half);
 130                 y = (y - n * pio2_1) - n * pio2_t;
 131 #if defined(__i386) && !defined(__amd64)
 132                 if (rp != fp_extended)
 133                         (void) __swapRP(rp);
 134 #endif
 135         } else {
 136                 if (ix >= 0x7f800000)
 137                         return (x / x); /* sin(Inf or NaN) is NaN */
 138                 hy = ((int *)&y)[HIWORD];
 139                 n = ((hy >> 20) & 0x7ff) - 1046;
 140                 ((int *)&w)[HIWORD] = (hy & 0xfffff) | 0x41600000;
 141                 ((int *)&w)[LOWORD] = ((int *)&y)[LOWORD];
 142                 n = __rem_pio2m(&w, &y, n, 1, 0, _TBL_ipio2_inf);
 143                 if (hy < 0) {
 144                         y = -y;
 145                         n = -n;
 146                 }
 147         }

 149         hy = ((int *)&y)[HIWORD] & ~0x80000000;
 150         if (hy < 0x3f800000) {          /* |y| < 2**-7 */
 151                 z = (y * T0) * (T1 + y * y);
 152                 f = ((n & 1) == 0)? (float)z : (float)(-one / z);
 153                 return (f);
 154         }
 155         z = y * y;
 156         w = ((P0 * y) * (P1 + z * (P2 + z)) * (P3 + z * (P4 + z))) *
 157             (P5 + z * (P6 + z * (P7 + z)));
 158         f = ((n & 1) == 0)? (float)w : (float)(-one / w);
 159         return (f);
 160 }
______unchanged_portion_omitted_
```

```
**********************************************************
    5524 Sun May 11 12:16:03 2014
new/usr/src/lib/libm/common/complex/k_cexp.c
**********************************************************
______unchanged_portion_omitted_
 110 invln2 = 1.44269504088896338700e+00,    /* 0x3ff71547, 0x652b82fe */
 111 P1 = 1.66666666666660190037e-01,        /* 0x3FC55555, 0x5555553E */
 112 P2 = -2.77777777770155933842e-03,       /* 0xBF66C16C, 0x16BEBD93 */
 113 P3 = 6.61375632143793436117e-05,        /* 0x3F11566A, 0xAF25DE2C */
 114 P4 = -1.65339022054652515390e-06,       /* 0xBEBBBD41, 0xC5D26BF1 */
 115 P5 = 4.13813679705723846039e-08;        /* 0x3E663769, 0x72BEA4D0 */
 116 /* INDENT ON */

 118 double
 119 __k_cexp(double x, int *n) {
 120         double hi = 0.0L, lo = 0.0L, c, t;
 120         double hi, lo, c, t;
 121         int k, xsb;
 122         unsigned hx, lx;

 124         hx = HI_WORD(x);        /* high word of x */
 125         lx = LO_WORD(x);        /* low word of x */
 126         xsb = (hx >> 31) & 1;   /* sign bit of x */
 127         hx &= 0x7fffffff;       /* high word of |x| */

 129         /* filter out non-finite argument */
 130         if (hx >= 0x40e86a00) { /* if |x| > 50000 */
 131                 if (hx >= 0x7ff00000) {
 132                         *n = 1;
 133                         if (((hx & 0xfffff) | lx) != 0)
 134                                 return (x + x); /* NaN */
 135                         else
 136                                 return ((xsb == 0) ? x : 0.0);
 137                                                         /* exp(+-inf)={inf,0} */
 138                 }
 139                 *n = (xsb == 0) ? 50000 : -50000;
 140                 return (one + ln2LO[1] * ln2LO[1]);     /* generate inexact */
 141         }

 143         *n = 0;
 144         /* argument reduction */
 145         if (hx > 0x3fd62e42) {  /* if  |x| > 0.5 ln2 */
 146                 if (hx < 0x3FF0A2B2) {  /* and |x| < 1.5 ln2 */
 147                         hi = x - ln2HI[xsb];
 148                         lo = ln2LO[xsb];
 149                         k = 1 - xsb - xsb;
 150                 } else {
 151                         k = (int) (invln2 * x + halF[xsb]);
 152                         t = k;
 153                         hi = x - t * ln2HI[0];
 154                                         /* t*ln2HI is exact for t<2**20 */
 155                         lo = t * ln2LO[0];
 156                 }
 157                 x = hi - lo;
 158                 *n = k;
 159         } else if (hx < 0x3e300000) {   /* when |x|<2**-28 */
 160                 return (one + x);
 161         } else
 162                 k = 0;

 164         /* x is now in primary range */
 165         t = x * x;
 166         c = x - t * (P1 + t * (P2 + t * (P3 + t * (P4 + t * P5))));
 167         if (k == 0)
 168                 return (one - ((x * c) / (c - 2.0) - x));
 169         else {
```



```
 170                 t = one - ((lo - (x * c) / (2.0 - c)) - hi);
 171                 if (k > 128) {
 172                         t *= two128;
 173                         *n = k - 128;
 174                 } else if (k > 0) {
 175                         HI_WORD(t) += (k << 20);
 176                         *n = 0;
 177                 }
 178                 return (t);
 179         }
 180 }
______unchanged_portion_omitted_
```

****
22566 Sun May 11 12:16:05 2014
new/usr/src/lib/libm/common/complex/k_clog_rl.c
****
_____unchanged_portion_omitted_

```
long double
__k_clog_rl(long double x, long double y, long double *er)
{
        long double t1, t2, t3, t4, tk, z, wh, w, zh, zk;
        int n, k, ix, iy, iz, nx, ny, nz, i;
        double dk;

#if !defined(__x86)
        int j;
        unsigned lx, ly;
#endif

        ix = HI_XWORD(x) & ~0x80000000;
        iy = HI_XWORD(y) & ~0x80000000;
        y = fabsl(y); x = fabsl(x);
        if (ix < iy || (ix < 0x7fff0000 && ix == iy && x < y)) {
                /* force x >= y */
                tk = x;  x = y;   y = tk;
                n = ix, ix = iy; iy = n;
        }
        *er = zero;
        nx = ix >> 16; ny = iy >> 16;
        if (nx >= 0x7fff) {     /* x or y is Inf or NaN */
                if (isinfl(x))
                        return (x);
                else if (isinfl(y))
                        return (y);
                else
                        return (x+y);
        }
/*
 * for tiny y:(double y < 2^-35, extended y < 2^-46, quad y < 2^-70)
 *
 *      log(sqrt(1 + y**2)) =  y**2 / 2 - y**4 / 8 + ...  =  y**2 / 2
 */
#if defined(__x86)
        if (x == 1.0L && ny < (0x3fff - 46)) {
#else
        if (x == 1.0L && ny < (0x3fff - 70)) {
#endif

                t2 = y * y;
                if (ny >= 8305) {       /* compute er = tail of t2 */
                        dk = (double) y;

#if defined(__x86)
                        ((unsigned *)&dk)[LOWORD] &= 0xfffe0000;
#endif

                        wh = (long double) dk;
                        *er = half * ((y - wh) * (y + wh) - (t2 - wh * wh));
                }
                return (half * t2);
        }
/*
 * x or y is subnormal or zero
 */
        if (nx == 0) {
                if (x == 0.0L)
                        return (-1.0L / x);
```



```
                else {
                        x *= two240;
                        y *= two240;
                        ix = HI_XWORD(x);
                        iy = HI_XWORD(y);
                        nx = (ix >> 16) - 240;
                        ny = (iy >> 16) - 240;
                        /* guard subnormal flush to 0 */
                        if (x == 0.0L)
                                return (-1.0L / x);
                }
        } else if (ny == 0) {   /* y subnormal, scale it */
                y *= two240;
                iy = HI_XWORD(y);
                ny = (iy >> 16) - 240;
        }
        n = nx - ny;
/*
 * When y is zero or when x >> y, i.e.,  n > 62, 78, 122 for DBLE,
 * EXTENDED, QUAD respectively,
 * log(x) = log(sqrt(x * x + y * y)) to 27 extra bits.
 */

#if defined(__x86)
        if (n > 78 || y == 0.0L) {
#else
        if (n > 122 || y == 0.0L) {
#endif

                XFSCALE(x, (0x3fff - (ix >> 16)));
                XFSCALE(x, 0x3fff - (ix >> 16));
                i = ((ix & 0xffff) + 0x100) >> 9;  /* 7.5 bits of x */
                zk = 1.0L + ((long double) i) * 0.0078125L;
                z = x - zk;
                dk = (double)z;

#if defined(__x86)
                ((unsigned *)&dk)[LOWORD] &= 0xfffe0000;
#endif

                zh = (long double)dk;
                k = i & 0x7f;   /* index of zk */
                n = nx - 0x3fff;
                *er = z - zh;
                if (i == 0x80) {        /* if zk = 2.0, adjust scaling */
                        n += 1;
                        zh *= 0.5L;  *er *= 0.5L;
                }
                w = k_log_NKzl(n, k, zh, er);
        } else {
/*
 * compute z = x*x + y*y
 */
                XFSCALE(x, (0x3fff - (ix >> 16)));
                XFSCALE(y, (0x3fff - n - (iy >> 16)));
                XFSCALE(x, 0x3fff - (ix >> 16));
                XFSCALE(y, 0x3fff - n - (iy >> 16));
                ix = (ix & 0xffff) | 0x3fff0000;
                iy = (iy & 0xffff) | (0x3fff0000 - (n << 16));
                nx -= 0x3fff;
                t1 = x * x; t2 = y * y;
                wh = x;

/* split x into correctly rounded half */
#if defined(__x86)
                ((unsigned *)&wh)[0] = 0;       /* 32 bits chopped */
```

```
#else
                lx = ((unsigned *)&wh)[2];      /* 56 rounded */
                j  = ((lx >> 24) + 1) >> 1;
                ((unsigned *)&wh)[2] = (j << 25);
                lx = ((unsigned *)&wh)[1];
                ly = lx + (j >> 7);
                ((unsigned *)&wh)[1] = ly;
                ((unsigned *)&wh)[0] += (ly == 0 && lx != 0);
                ((unsigned *)&wh)[3] = 0;
#endif

                z = t1+t2;
/*
 * higher precision simulation x*x = t1 + t3, y*y = t2 + t4
 */
                tk = wh - x;
                t3 = tk * tk - (two * wh * tk - (wh * wh - t1));
                wh = y;

/* split y into correctly rounded half */
#if defined(__x86)
                ((unsigned *)&wh)[0] = 0;       /* 32 bits chopped */
#else
                ly = ((unsigned *)&wh)[2];      /* 56 bits rounded */
                j  = ((ly >> 24) + 1) >> 1;
                ((unsigned *)&wh)[2] = (j << 25);
                lx = ((unsigned *)&wh)[1];
                ly = lx + (j >> 7);
                ((unsigned *)&wh)[1] = ly;
                ((unsigned *)&wh)[0] += (ly == 0 && lx != 0);
                ((unsigned *)&wh)[3] = 0;
#endif

                tk = wh - y;
                t4 = tk * tk - (two * wh * tk - (wh * wh - t2));
/*
 * find zk matches z to 7.5 bits
 */
                iz = HI_XWORD(z);
                k = ((iz & 0xffff) + 0x100) >> 9;  /* 7.5 bits of x */
                nz = (iz >> 16) - 0x3fff + (k >> 7);
                k &= 0x7f;
                zk = 1.0L + ((long double) k) * 0.0078125L;
                if (nz == 1) zk += zk;
                else if (nz == 2) zk *= 4.0L;
                else if (nz == 3) zk *= 8.0L;
/*
 * order t1, t2, t3, t4 according to their size
 */
                if (t2 >= fabsl(t3)) {
                        if (fabsl(t3) < fabsl(t4)) {
                                wh = t3;  t3 = t4; t4 = wh;
                        }
                } else {
                        wh = t2; t2 = t3; t3 = wh;
                }
/*
 * higher precision simulation: x * x + y * y = t1 + t2 + t3 + t4
 * = zk(7 bits) + zh(24 bits) + *er(tail) and call k_log_NKz
 */
                tk = t1 - zk;
                zh = ((tk + t2) + t3) + t4;

/* split zh into correctly rounded half */
#if defined(__x86)
                ((unsigned *)&zh)[0] = 0;
```



```
#else
                ly = ((unsigned *)&zh)[2];
                j  = ((ly >> 24) + 1) >> 1;
                ((unsigned *)&zh)[2] = (j << 25);
                lx = ((unsigned *)&zh)[1];
                ly = lx + (j >> 7);
                ((unsigned *)&zh)[1] = ly;
                ((unsigned *)&zh)[0] += (ly == 0 && lx != 0);
                ((unsigned *)&zh)[3] = 0;
#endif

                w = fabsl(zh);
                if (w >= fabsl(t2))
{
                        *er = (((tk - zh) + t2) + t3) + t4;
}
______unchanged_portion_omitted_
```

```
**********************************************************
   21518 Sun May 11 12:16:06 2014
new/usr/src/lib/libm/common/m9x/__fex_hdlr.c
**********************************************************
______unchanged_portion_omitted_

 375 #elif defined(__x86)

 377 #if defined(__amd64)
 378 #define test_sse_hw     1
 379 #else
 380 extern int _sse_hw;
 381 #define test_sse_hw     _sse_hw
 382 #endif

 384 #if !defined(REG_PC)
 385 #define REG_PC  EIP
 386 #endif

 388 /*
 389 *  If a handling mode is in effect, apply it; otherwise invoke the
 390 *  saved handler
 391 */
 392 static void
 393 __fex_hdlr(int sig, siginfo_t *sip, ucontext_t *uap)
 394 {
 395         struct fex_handler_data *thr_handlers;
 396         struct sigaction        act;
 397         void                    (*handler)() = NULL, (*simd_handler[4])();
 397         void                    (*handler)(), (*simd_handler[4])();
 398         int                     mode, simd_mode[4], i, len, accrued, *ap;
 399         unsigned int            cwsw, oldcwsw, mxcsr, oldmxcsr;
 400         enum fex_exception      e, simd_e[4];
 401         fex_info_t              info, simd_info[4];
 402         unsigned long           addr;
 403         siginfo_t               osip = *sip;
 404         sseinst_t               inst;

 406         /* check for an exception caused by an SSE instruction */
 407         if (!(uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state.status & 0x80)) {
 408                 len = __fex_parse_sse(uap, &inst);
 409                 if (len == 0)
 410                         goto not_ieee;

 412                 /* disable all traps and clear flags */
 413                 __fenv_getcwsw(&oldcwsw);
 414                 cwsw = (oldcwsw & ~0x3f) | 0x003f0000;
 415                 __fenv_setcwsw(&cwsw);
 416                 __fenv_getmxcsr(&oldmxcsr);
 417                 mxcsr = (oldmxcsr & ~0x3f) | 0x1f80;
 418                 __fenv_setmxcsr(&mxcsr);

 420                 if ((int)inst.op & SIMD) {
 421                         __fex_get_simd_op(uap, &inst, simd_e, simd_info);

 423                         thr_handlers = __fex_get_thr_handlers();
 424                         addr = (unsigned long)uap->uc_mcontext.gregs[REG_PC];
 425                         accrued = uap->uc_mcontext.fpregs.fp_reg_set.
 426                             fpchip_state.mxcsr;

 428                         e = (enum fex_exception)-1;
 429                         mode = FEX_NONSTOP;
 430                         for (i = 0; i < 4; i++) {
 431                                 if ((int)simd_e[i] < 0)
 432                                         continue;
```



```
 434                                 e = simd_e[i];
 435                                 simd_mode[i] = FEX_NOHANDLER;
 436                                 simd_handler[i] = oact.sa_handler;
 437                                 if (thr_handlers &&
 438                                     thr_handlers[(int)e].__mode !=
 439                                     FEX_NOHANDLER) {
 440                                         simd_mode[i] =
 441                                             thr_handlers[(int)e].__mode;
 442                                         simd_handler[i] =
 443                                             thr_handlers[(int)e].__handler;
 444                                 }
 445                                 accrued &= ~te_bit[(int)e];
 446                                 switch (simd_mode[i]) {
 447                                 case FEX_ABORT:
 448                                         mode = FEX_ABORT;
 449                                         break;
 450                                 case FEX_SIGNAL:
 451                                         if (mode != FEX_ABORT)
 452                                                 mode = FEX_SIGNAL;
 453                                         handler = simd_handler[i];
 454                                         break;
 455                                 case FEX_NOHANDLER:
 456                                         if (mode != FEX_ABORT && mode !=
 457                                             FEX_SIGNAL)
 458                                                 mode = FEX_NOHANDLER;
 459                                         break;
 460                                 }
 461                         }
 462                         if (e == (enum fex_exception)-1) {
 463                                 __fenv_setcwsw(&oldcwsw);
 464                                 __fenv_setmxcsr(&oldmxcsr);
 465                                 goto not_ieee;
 466                         }
 467                         accrued |= uap->uc_mcontext.fpregs.fp_reg_set.
 468                             fpchip_state.status;
 469                         ap = __fex_accrued();
 470                         accrued |= *ap;
 471                         accrued &= 0x3d;

 473                         for (i = 0; i < 4; i++) {
 474                                 if ((int)simd_e[i] < 0)
 475                                         continue;

 477                                 __fex_mklog(uap, (char *)addr, accrued,
 478                                     simd_e[i], simd_mode[i],
 479                                     (void *)simd_handler[i]);
 480                         }

 482                         if (mode == FEX_NOHANDLER) {
 483                                 __fenv_setcwsw(&oldcwsw);
 484                                 __fenv_setmxcsr(&oldmxcsr);
 485                                 goto not_ieee;
 486                         } else if (mode == FEX_ABORT) {
 487                                 abort();
 488                         } else if (mode == FEX_SIGNAL) {
 489                                 __fenv_setcwsw(&oldcwsw);
 490                                 __fenv_setmxcsr(&oldmxcsr);
 491                                 handler(sig, &osip, uap);
 492                                 return;
 493                         }

 495                         *ap = 0;
 496                         for (i = 0; i < 4; i++) {
 497                                 if ((int)simd_e[i] < 0)
 498                                         continue;
```

```
500                                 if (simd_mode[i] == FEX_CUSTOM) {
501                                         handler(1 << (int)simd_e[i],
502                                             &simd_info[i]);
503                                         __fenv_setcwsw(&cwsw);
504                                         __fenv_setmxcsr(&mxcsr);
505                                 }
506                         }

508                         __fex_st_simd_result(uap, &inst, simd_e, simd_info);
509                         for (i = 0; i < 4; i++) {
510                                 if ((int)simd_e[i] < 0)
511                                         continue;

513                                 accrued |= simd_info[i].flags;
514                         }

516                         if ((int)inst.op & INTREG) {
517                                 /* set MMX mode */
518 #if defined(__amd64)
519                                 uap->uc_mcontext.fpregs.fp_reg_set.
520                                     fpchip_state.sw &= ~0x3800;
521                                 uap->uc_mcontext.fpregs.fp_reg_set.
522                                     fpchip_state.fctw = 0;
523 #else
524                                 uap->uc_mcontext.fpregs.fp_reg_set.
525                                     fpchip_state.state[1] &= ~0x3800;
526                                 uap->uc_mcontext.fpregs.fp_reg_set.
527                                     fpchip_state.state[2] = 0;
528 #endif
529                         }
530                 } else {
531                         e = __fex_get_sse_op(uap, &inst, &info);
532                         if ((int)e < 0) {
533                                 __fenv_setcwsw(&oldcwsw);
534                                 __fenv_setmxcsr(&oldmxcsr);
535                                 goto not_ieee;
536                         }

538                         mode = FEX_NOHANDLER;
539                         handler = oact.sa_handler;
540                         thr_handlers = __fex_get_thr_handlers();
541                         if (thr_handlers && thr_handlers[(int)e].__mode !=
542                             FEX_NOHANDLER) {
543                                 mode = thr_handlers[(int)e].__mode;
544                                 handler = thr_handlers[(int)e].__handler;
545                         }

547                         addr = (unsigned long)uap->uc_mcontext.gregs[REG_PC];
548                         accrued = uap->uc_mcontext.fpregs.fp_reg_set.
549                             fpchip_state.mxcsr & ~te_bit[(int)e];
550                         accrued |= uap->uc_mcontext.fpregs.fp_reg_set.
551                             fpchip_state.status;
552                         ap = __fex_accrued();
553                         accrued |= *ap;
554                         accrued &= 0x3d;
555                         __fex_mklog(uap, (char *)addr, accrued, e, mode,
556                             (void *)handler);

558                         if (mode == FEX_NOHANDLER) {
559                                 __fenv_setcwsw(&oldcwsw);
560                                 __fenv_setmxcsr(&oldmxcsr);
561                                 goto not_ieee;
562                         } else if (mode == FEX_ABORT) {
563                                 abort();
564                         } else if (mode == FEX_SIGNAL) {
565                                 __fenv_setcwsw(&oldcwsw);
```



```
566                                 __fenv_setmxcsr(&oldmxcsr);
567                                 handler(sig, &osip, uap);
568                                 return;
569                         } else if (mode == FEX_CUSTOM) {
570                                 *ap = 0;
571                                 if (addr >= (unsigned long)feraiseexcept &&
572                                     addr < (unsigned long)fetestexcept ) {
573                                         info.op = fex_other;
574                                         info.op1.type = info.op2.type =
575                                             info.res.type = fex_nodata;
576                                 }
577                                 handler(1 << (int)e, &info);
578                                 __fenv_setcwsw(&cwsw);
579                                 __fenv_setmxcsr(&mxcsr);
580                         }

582                         __fex_st_sse_result(uap, &inst, e, &info);
583                         accrued |= info.flags;

585 #if defined(__amd64)
586                         /*
587                          * In 64-bit mode, the 32-bit convert-to-integer
588                          * instructions zero the upper 32 bits of the
589                          * destination.  (We do this here and not in
590                          * __fex_st_sse_result because __fex_st_sse_result
591                          * can be called from __fex_st_simd_result, too.)
592                          */
593                         if (inst.op == cvtss2si || inst.op == cvttss2si ||
594                             inst.op == cvtsd2si || inst.op == cvttsd2si)
595                                 inst.op1->i[1] = 0;
596 #endif
597                 }

599                 /* advance the pc past the SSE instruction */
600                 uap->uc_mcontext.gregs[REG_PC] += len;
601                 goto update_state;
602         }

604         /* determine which exception occurred */
605         __fex_get_x86_exc(sip, uap);
606         switch (sip->si_code) {
607         case FPE_FLTDIV:
608                 e = fex_division;
609                 break;
610         case FPE_FLTOVF:
611                 e = fex_overflow;
612                 break;
613         case FPE_FLTUND:
614                 e = fex_underflow;
615                 break;
616         case FPE_FLTRES:
617                 e = fex_inexact;
618                 break;
619         case FPE_FLTINV:
620                 if ((int)(e = __fex_get_invalid_type(sip, uap)) < 0)
621                         goto not_ieee;
622                 break;
623         default:
624                 /* not an IEEE exception */
625                 goto not_ieee;
626         }

628         /* get the handling mode */
629         mode = FEX_NOHANDLER;
630         handler = oact.sa_handler; /* for log; just looking, no need to lock */
631         thr_handlers = __fex_get_thr_handlers();
```

```
632         if (thr_handlers && thr_handlers[(int)e].__mode != FEX_NOHANDLER) {
633                 mode = thr_handlers[(int)e].__mode;
634                 handler = thr_handlers[(int)e].__handler;
635         }

637         /* make an entry in the log of retro. diag. if need be */
638 #if defined(__amd64)
639         addr = (unsigned long)uap->uc_mcontext.fpregs.fp_reg_set.
640             fpchip_state.rip;
641 #else
642         addr = (unsigned long)uap->uc_mcontext.fpregs.fp_reg_set.
643             fpchip_state.state[3];
644 #endif
645         accrued = uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state.status &
646             ~te_bit[(int)e];
647         if (test_sse_hw)
648                 accrued |= uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state.
649                     mxcsr;
650         ap = __fex_accrued();
651         accrued |= *ap;
652         accrued &= 0x3d;
653         __fex_mklog(uap, (char *)addr, accrued, e, mode, (void *)handler);

655         /* handle the exception based on the mode */
656         if (mode == FEX_NOHANDLER)
657                 goto not_ieee;
658         else if (mode == FEX_ABORT)
659                 abort();
660         else if (mode == FEX_SIGNAL) {
661                 handler(sig, &osip, uap);
662                 return;
663         }

665         /* disable all traps and clear flags */
666         __fenv_getcwsw(&cwsw);
667         cwsw = (cwsw & ~0x3f) | 0x003f0000;
668         __fenv_setcwsw(&cwsw);
669         if (test_sse_hw) {
670                 __fenv_getmxcsr(&mxcsr);
671                 mxcsr = (mxcsr & ~0x3f) | 0x1f80;
672                 __fenv_setmxcsr(&mxcsr);
673         }
674         *ap = 0;

676         /* decode the operation */
677         __fex_get_op(sip, uap, &info);

679         /* if a custom mode handler is installed, invoke it */
680         if (mode == FEX_CUSTOM) {
681                 /* if we got here from feraiseexcept, pass dummy info */
682                 if (addr >= (unsigned long)feraiseexcept &&
683                     addr < (unsigned long)fetestexcept ) {
684                         info.op = fex_other;
685                         info.op1.type = info.op2.type = info.res.type =
686                             fex_nodata;
687                 }

689                 handler(1 << (int)e, &info);

691                 /* restore modes in case the user's handler changed them */
692                 __fenv_setcwsw(&cwsw);
693                 if (test_sse_hw)
694                         __fenv_setmxcsr(&mxcsr);
695         }

697         /* stuff the result */
```



```
698         __fex_st_result(sip, uap, &info);
699         accrued |= info.flags;

701 update_state:
702         accrued &= 0x3d;
703         i = __fex_te_needed(thr_handlers, accrued);
704         *ap = accrued & i;
705 #if defined(__amd64)
706         uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state.sw &= ~0x3d;
707         uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state.sw |= (accrued & ~i);
708         uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state.cw |= 0x3d;
709         uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state.cw &= ~i;
710 #else
711         uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state.state[1] &= ~0x3d;
712         uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state.state[1] |=
713             (accrued & ~i);
714         uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state.state[0] |= 0x3d;
715         uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state.state[0] &= ~i;
716 #endif
717         if (test_sse_hw) {
718                 uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state.mxcsr &= ~0x3d;
719                 uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state.mxcsr |=
720                     0x1e80 | (accrued & ~i);
721                 uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state.mxcsr &=
722                     ~(i << 7);
723         }
724         return;

726 not_ieee:
727         /* revert to the saved handler (if any) */
728         mutex_lock(&hdlr_lock);
729         act = oact;
730         mutex_unlock(&hdlr_lock);
731         switch ((unsigned long)act.sa_handler) {
732         case (unsigned long)SIG_DFL:
733                 /* simulate trap with no handler installed */
734                 sigaction(SIGFPE, &act, NULL);
735                 kill(getpid(), SIGFPE);
736                 break;
737 #if !defined(__lint)
738         case (unsigned long)SIG_IGN:
739                 break;
740 #endif
741         default:
742                 act.sa_handler(sig, &osip, uap);
743         }
744 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   36583 Sun May 11 12:16:07 2014
new/usr/src/lib/libm/common/m9x/__fex_i386.c
**********************************************************
______unchanged_portion_omitted_

1227 /* scale factors for exponent wrapping */
1228 static const float
1229         fun = 7.922816251e+28f, /* 2^96 */
1230         fov = 1.262177448e-29f; /* 2^-96 */
1231 static const double
1232         dun = 1.5525180923007089350e+231,        /* 2^768 */
1233         dov = 6.4411487695971333080e-232;        /* 2^-768 */

1235 /*
1236 *  Store the specified result; if no result is given but the exception
1237 *  is underflow or overflow, use the default trapped result
1238 */
1239 void
1240 __fex_st_result(siginfo_t *sip, ucontext_t *uap, fex_info_t *info)
1241 {
1242         fex_numeric_t   r;
1243         unsigned long           ex, op, ea, stack;

1245         /* get the exception type, opcode, and data address */
1246         ex = sip->si_code;
1247 #if defined(__amd64)
1248         op = uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state.fop >> 16;
1249         ea = uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state.rdp; /*???*/
1250 #else
1251         op = uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state.state[OP] >> 16;
1252         ea = uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state.state[EA];
1253 #endif

1255         /* if the instruction is a compare, set the condition codes
1256            to unordered and update the stack */
1257         switch (op & 0x7f8) {
1258         case 0x010:
1259         case 0x050:
1260         case 0x090:
1261         case 0x0d0:
1262         case 0x210:
1263         case 0x250:
1264         case 0x290:
1265         case 0x410:
1266         case 0x450:
1267         case 0x490:
1268         case 0x4d0:
1269         case 0x5e0:
1270         case 0x610:
1271         case 0x650:
1272         case 0x690:
1273                 /* f[u]com */
1274 #if defined(__amd64)
1275                 uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state.sw |= 0x4500;
1276 #else
1277                 uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state.state[SW] |= 0x4
1278 #endif
1279                 return;

1281         case 0x018:
1282         case 0x058:
1283         case 0x098:
1284         case 0x0d8:
1285         case 0x218:
1286         case 0x258:
```



```
1287         case 0x298:
1288         case 0x418:
1289         case 0x458:
1290         case 0x498:
1291         case 0x4d8:
1292         case 0x5e8:
1293         case 0x618:
1294         case 0x658:
1295         case 0x698:
1296         case 0x6d0:
1297                 /* f[u]comp */
1298 #if defined(__amd64)
1299                 uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state.sw |= 0x4500;
1300 #else
1301                 uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state.state[SW] |= 0x4
1302 #endif
1303                 pop(uap);
1304                 return;

1306         case 0x2e8:
1307         case 0x6d8:
1308                 /* f[u]compp */
1309 #if defined(__amd64)
1310                 uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state.sw |= 0x4500;
1311 #else
1312                 uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state.state[SW] |= 0x4
1313 #endif
1314                 pop(uap);
1315                 pop(uap);
1316                 return;

1318         case 0x1e0:
1319                 if (op == 0x1e4) { /* ftst */
1320 #if defined(__amd64)
1321                         uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state.sw |= 0x
1322 #else
1323                         uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state.state[SW
1324 #endif
1325                         return;
1326                 }
1327                 break;

1329         case 0x3e8:
1330         case 0x3f0:
1331                 /* f[u]comi */
1332 #if defined(__amd64)
1333                 uap->uc_mcontext.gregs[REG_PS] |= 0x45;
1334 #else
1335                 uap->uc_mcontext.gregs[EFL] |= 0x45;
1336 #endif
1337                 return;

1339         case 0x7e8:
1340         case 0x7f0:
1341                 /* f[u]comip */
1342 #if defined(__amd64)
1343                 uap->uc_mcontext.gregs[REG_PS] |= 0x45;
1344 #else
1345                 uap->uc_mcontext.gregs[EFL] |= 0x45;
1346 #endif
1347                 pop(uap);
1348                 return;
1349         }

1351         /* if there is no result available and the exception is overflow
1352            or underflow, use the wrapped result */
```

```
1353         r = info->res;
1354         if (r.type == fex_nodata) {
1355                 if (ex == FPE_FLTOVF || ex == FPE_FLTUND) {
1356                         /* for store instructions, do the scaling and store */
1357                         switch (op & 0x7f8) {
1358                         case 0x110:
1359                         case 0x118:
1360                         case 0x150:
1361                         case 0x158:
1362                         case 0x190:
1363                         case 0x198:
1364                                 if (!ea)
1365                                         return;
1366                                 if (ex == FPE_FLTOVF)
1367                                         *(float *)ea = (fpreg(uap, 0) * fov) * f
1368                                 else
1369                                         *(float *)ea = (fpreg(uap, 0) * fun) * f
1370                                 if ((op & 8) != 0)
1371                                         pop(uap);
1372                                 break;

1374                         case 0x510:
1375                         case 0x518:
1376                         case 0x550:
1377                         case 0x558:
1378                         case 0x590:
1379                         case 0x598:
1380                                 if (!ea)
1381                                         return;
1382                                 if (ex == FPE_FLTOVF)
1383                                         *(double *)ea = (fpreg(uap, 0) * dov) *
1384                                 else
1385                                         *(double *)ea = (fpreg(uap, 0) * dun) *
1386                                 if ((op & 8) != 0)
1387                                         pop(uap);
1388                                 break;
1389                         }
1390                 }
1391 #ifdef DEBUG
1392                 else if (ex != FPE_FLTRES)
1393                         printf( "No result supplied, stack may be hosed\n" );
1394 #endif
1395                 return;
1396         }

1398         /* otherwise convert the supplied result to the correct type,
1399            put it in the destination, and update the stack as need be */

1401         /* store instructions */
1402         switch (op & 0x7f8) {
1403         case 0x110:
1404         case 0x118:
1405         case 0x150:
1406         case 0x158:
1407         case 0x190:
1408         case 0x198:
1409                 if (!ea)
1410                         return;
1411                 switch (r.type) {
1412                 case fex_int:
1413                         *(float *)ea = (float) r.val.i;
1414                         break;

1416                 case fex_llong:
1417                         *(float *)ea = (float) r.val.l;
1418                         break;
```



```
1420                 case fex_float:
1421                         *(float *)ea = r.val.f;
1422                         break;

1424                 case fex_double:
1425                         *(float *)ea = (float) r.val.d;
1426                         break;

1428                 case fex_ldouble:
1429                         *(float *)ea = (float) r.val.q;
1430                         break;

1432                 default:
1433                         break;
1434 #endif /* ! codereview */
1435                 }
1436                 if (ex != FPE_FLTRES && (op & 8) != 0)
1437                         pop(uap);
1438                 return;

1440         case 0x310:
1441         case 0x318:
1442         case 0x350:
1443         case 0x358:
1444         case 0x390:
1445         case 0x398:
1446                 if (!ea)
1447                         return;
1448                 switch (r.type) {
1449                 case fex_int:
1450                         *(int *)ea = r.val.i;
1451                         break;

1453                 case fex_llong:
1454                         *(int *)ea = (int) r.val.l;
1455                         break;

1457                 case fex_float:
1458                         *(int *)ea = (int) r.val.f;
1459                         break;

1461                 case fex_double:
1462                         *(int *)ea = (int) r.val.d;
1463                         break;

1465                 case fex_ldouble:
1466                         *(int *)ea = (int) r.val.q;
1467                         break;

1469                 default:
1470                         break;
1471 #endif /* ! codereview */
1472                 }
1473                 if (ex != FPE_FLTRES && (op & 8) != 0)
1474                         pop(uap);
1475                 return;

1477         case 0x510:
1478         case 0x518:
1479         case 0x550:
1480         case 0x558:
1481         case 0x590:
1482         case 0x598:
1483                 if (!ea)
1484                         return;
```

```
1485                switch (r.type) {
1486                case fex_int:
1487                        *(double *)ea = (double) r.val.i;
1488                        break;
1489
1490                case fex_llong:
1491                        *(double *)ea = (double) r.val.l;
1492                        break;
1493
1494                case fex_float:
1495                        *(double *)ea = (double) r.val.f;
1496                        break;
1497
1498                case fex_double:
1499                        *(double *)ea = r.val.d;
1500                        break;
1501
1502                case fex_ldouble:
1503                        *(double *)ea = (double) r.val.q;
1504                        break;
1505
1506                default:
1507                        break;
1508 #endif /* ! codereview */
1509                }
1510                if (ex != FPE_FLTRES && (op & 8) != 0)
1511                        pop(uap);
1512                return;
1513
1514        case 0x710:
1515        case 0x718:
1516        case 0x750:
1517        case 0x758:
1518        case 0x790:
1519        case 0x798:
1520                if (!ea)
1521                        return;
1522                switch (r.type) {
1523                case fex_int:
1524                        *(short *)ea = (short) r.val.i;
1525                        break;
1526
1527                case fex_llong:
1528                        *(short *)ea = (short) r.val.l;
1529                        break;
1530
1531                case fex_float:
1532                        *(short *)ea = (short) r.val.f;
1533                        break;
1534
1535                case fex_double:
1536                        *(short *)ea = (short) r.val.d;
1537                        break;
1538
1539                case fex_ldouble:
1540                        *(short *)ea = (short) r.val.q;
1541                        break;
1542
1543                default:
1544                        break;
1545 #endif /* ! codereview */
1546                }
1547                if (ex != FPE_FLTRES && (op & 8) != 0)
1548                        pop(uap);
1549                return;
```



```
1551        case 0x730:
1552        case 0x770:
1553        case 0x7b0:
1554                /* fbstp; don't bother */
1555                if (ea && ex != FPE_FLTRES)
1556                        pop(uap);
1557                return;
1558
1559        case 0x738:
1560        case 0x778:
1561        case 0x7b8:
1562                if (!ea)
1563                        return;
1564                switch (r.type) {
1565                case fex_int:
1566                        *(long long *)ea = (long long) r.val.i;
1567                        break;
1568
1569                case fex_llong:
1570                        *(long long *)ea = r.val.l;
1571                        break;
1572
1573                case fex_float:
1574                        *(long long *)ea = (long long) r.val.f;
1575                        break;
1576
1577                case fex_double:
1578                        *(long long *)ea = (long long) r.val.d;
1579                        break;
1580
1581                case fex_ldouble:
1582                        *(long long *)ea = (long long) r.val.q;
1583                        break;
1584
1585                default:
1586                        break;
1587 #endif /* ! codereview */
1588                }
1589                if (ex != FPE_FLTRES)
1590                        pop(uap);
1591                return;
1592        }
1593
1594        /* for all other instructions, the result goes into a register */
1595        switch (r.type) {
1596        case fex_int:
1597                r.val.q = (long double) r.val.i;
1598                break;
1599
1600        case fex_llong:
1601                r.val.q = (long double) r.val.l;
1602                break;
1603
1604        case fex_float:
1605                r.val.q = (long double) r.val.f;
1606                break;
1607
1608        case fex_double:
1609                r.val.q = (long double) r.val.d;
1610                break;
1611
1612        default:
1613 #endif /* ! codereview */
1614                break;
1615        }
```

```
1617         /* for load instructions, push the result onto the stack */
1618         switch (op & 0x7f8) {
1619         case 0x100:
1620         case 0x140:
1621         case 0x180:
1622         case 0x500:
1623         case 0x540:
1624         case 0x580:
1625                 if (ea)
1626                         push(r.val.q, uap);
1627                 return;
1628         }

1630         /* for all other instructions, if the exception is overflow,
1631            underflow, or inexact, the stack has already been updated */
1632         stack = (ex == FPE_FLTOVF || ex == FPE_FLTUND || ex == FPE_FLTRES);
1633         switch (op & 0x7f8) {
1634         case 0x1f0: /* oddballs */
1635                 switch (op) {
1636                 case 0x1f1: /* fyl2x */
1637                 case 0x1f3: /* fpatan */
1638                 case 0x1f9: /* fyl2xp1 */
1639                         /* pop the stack, leaving the result in st */
1640                         if (!stack)
1641                                 pop(uap);
1642                         fpreg(uap, 0) = r.val.q;
1643                         return;

1645                 case 0x1f2: /* fpatan */
1646                         /* fptan pushes 1.0 afterward */
1647                         if (stack)
1648                                 fpreg(uap, 1) = r.val.q;
1649                         else {
1650                                 fpreg(uap, 0) = r.val.q;
1651                                 push(1.0L, uap);
1652                         }
1653                         return;

1655                 case 0x1f4: /* fxtract */
1656                 case 0x1fb: /* fsincos */
1657                         /* leave the supplied result in st */
1658                         if (stack)
1659                                 fpreg(uap, 0) = r.val.q;
1660                         else {
1661                                 fpreg(uap, 0) = 0.0; /* punt */
1662                                 push(r.val.q, uap);
1663                         }
1664                         return;
1665                 }

1667                 /* all others leave the stack alone and the result in st */
1668                 fpreg(uap, 0) = r.val.q;
1669                 return;

1671         case 0x4c0:
1672         case 0x4c8:
1673         case 0x4e0:
1674         case 0x4e8:
1675         case 0x4f0:
1676         case 0x4f8:
1677                 fpreg(uap, op & 7) = r.val.q;
1678                 return;

1680         case 0x6c0:
1681         case 0x6c8:
1682         case 0x6e0:
```



```
1683         case 0x6e8:
1684         case 0x6f0:
1685         case 0x6f8:
1686                 /* stack is popped afterward */
1687                 if (stack)
1688                         fpreg(uap, (op - 1) & 7) = r.val.q;
1689                 else {
1690                         fpreg(uap, op & 7) = r.val.q;
1691                         pop(uap);
1692                 }
1693                 return;

1695         default:
1696                 fpreg(uap, 0) = r.val.q;
1697                 return;
1698         }
1699 }
```

```
**********************************************************
   21370 Sun May 11 12:16:08 2014
new/usr/src/lib/libm/common/m9x/__fex_sparc.c
**********************************************************
______unchanged_portion_omitted_

 472 /*
 473 *  Store the specified result; if no result is given but the exception
 474 *  is underflow or overflow, supply the default trapped result
 475 */
 476 void
 477 __fex_st_result(siginfo_t *sip, ucontext_t *uap, fex_info_t *info)
 478 {
 479         unsigned                instr, opf, rs1, rs2, rd;
 480         long double             qscl;
 481         double                  dscl;
 482         float                   fscl;

 484         /* parse the instruction which caused the exception */
 485         instr = uap->uc_mcontext.fpregs.fpu_q->FQu.fpq.fpq_instr;
 486         opf = (instr >> 5) & 0x1ff;
 487         rs1 = (instr >> 14) & 0x1f;
 488         rs2 = instr & 0x1f;
 489         rd = (instr >> 25) & 0x1f;

 491         /* if the instruction is a compare, just set fcc to unordered */
 492         if (((instr >> 19) & 0x183f) == 0x1035) {
 493                 if (rd == 0)
 494                         uap->uc_mcontext.fpregs.fpu_fsr |= 0xc00;
 495                 else {
 496 #ifdef __sparcv9
 497                         uap->uc_mcontext.fpregs.fpu_fsr |= (3l << ((rd << 1) + 3
 498 #else
 499                         ((prxregset_t*)uap->uc_mcontext.xrs.xrs_ptr)->pr_un.pr_v
 500 #endif
 501                 }
 502                 return;
 503         }

 505         /* if there is no result available, try to generate the untrapped
 506            default */
 507         if (info->res.type == fex_nodata) {
 508                 /* set scale factors for exponent wrapping */
 509                 switch (sip->si_code) {
 510                 case FPE_FLTOVF:
 511                         fscl = 1.262177448e-29f;        /* 2^-96 */
 512                         dscl = 6.441148769597133308e-232;       /* 2^-768 */
 513                         qscl = 8.778357852076208839765066529179033145e-3700l;/*
 514                         break;

 516                 case FPE_FLTUND:
 517                         fscl = 7.922816251e+28f;        /* 2^96 */
 518                         dscl = 1.552518092300708935e+231;       /* 2^768 */
 519                         qscl = 1.139165225263043370845938579315932009e+3699l;/*
 520                         break;

 522                 default:
 523                         /* user may have blown away the default result by mistak
 524                            so try to regenerate it */
 525                         (void) __fex_get_op(sip, uap, info);
 526                         if (info->res.type != fex_nodata)
 527                                 goto stuff;
 528                         /* couldn't do it */
 529                         return;
 530                 }
```



```
 532                 /* get the operands */
 533                 switch (opf & 3) {
 534                 case 1: /* single */
 535                         info->op1.val.f = *(float*)FPreg(rs1);
 536                         info->op2.val.f = *(float*)FPreg(rs2);
 537                         break;

 539                 case 2: /* double */
 540                         info->op1.val.d = *(double*)FPREG(rs1);
 541                         info->op2.val.d = *(double*)FPREG(rs2);
 542                         break;

 544                 case 3: /* quad */
 545                         info->op1.val.q = *(long double*)FPREG(rs1);
 546                         info->op2.val.q = *(long double*)FPREG(rs2);
 547                         break;
 548                 }

 550                 /* generate the wrapped result */
 551                 switch (opf) {
 552                 case 0x41: /* add single */
 553                         info->res.type = fex_float;
 554                         info->res.val.f = fscl * ( fscl * info->op1.val.f +
 555                                 fscl * info->op2.val.f );
 556                         break;

 558                 case 0x42: /* add double */
 559                         info->res.type = fex_double;
 560                         info->res.val.d = dscl * ( dscl * info->op1.val.d +
 561                                 dscl * info->op2.val.d );
 562                         break;

 564                 case 0x43: /* add quad */
 565                         info->res.type = fex_ldouble;
 566                         info->res.val.q = qscl * ( qscl * info->op1.val.q +
 567                                 qscl * info->op2.val.q );
 568                         break;

 570                 case 0x45: /* subtract single */
 571                         info->res.type = fex_float;
 572                         info->res.val.f = fscl * ( fscl * info->op1.val.f -
 573                                 fscl * info->op2.val.f );
 574                         break;

 576                 case 0x46: /* subtract double */
 577                         info->res.type = fex_double;
 578                         info->res.val.d = dscl * ( dscl * info->op1.val.d -
 579                                 dscl * info->op2.val.d );
 580                         break;

 582                 case 0x47: /* subtract quad */
 583                         info->res.type = fex_ldouble;
 584                         info->res.val.q = qscl * ( qscl * info->op1.val.q -
 585                                 qscl * info->op2.val.q );
 586                         break;

 588                 case 0x49: /* multiply single */
 589                         info->res.type = fex_float;
 590                         info->res.val.f = ( fscl * info->op1.val.f ) *
 591                                 ( fscl * info->op2.val.f );
 592                         break;

 594                 case 0x4a: /* multiply double */
 595                         info->res.type = fex_double;
 596                         info->res.val.d = ( dscl * info->op1.val.d ) *
 597                                 ( dscl * info->op2.val.d );
```

```
598                         break;
600                 case 0x4b: /* multiply quad */
601                         info->res.type = fex_ldouble;
602                         info->res.val.q = ( qscl * info->op1.val.q ) *
603                                 ( qscl * info->op2.val.q );
604                         break;
606                 case 0x4d: /* divide single */
607                         info->res.type = fex_float;
608                         info->res.val.f = ( fscl * info->op1.val.f ) /
609                                 ( info->op2.val.f / fscl );
610                         break;
612                 case 0x4e: /* divide double */
613                         info->res.type = fex_double;
614                         info->res.val.d = ( dscl * info->op1.val.d ) /
615                                 ( info->op2.val.d / dscl );
616                         break;
618                 case 0x4f: /* divide quad */
619                         info->res.type = fex_ldouble;
620                         info->res.val.q = ( qscl * info->op1.val.q ) /
621                                 ( info->op2.val.q / qscl );
622                         break;
624                 case 0xc6: /* convert double to single */
625                         info->res.type = fex_float;
626                         info->res.val.f = (float) ( fscl * ( fscl * info->op1.va
627                         break;
629                 case 0xc7: /* convert quad to single */
630                         info->res.type = fex_float;
631                         info->res.val.f = (float) ( fscl * ( fscl * info->op1.va
632                         break;
634                 case 0xcb: /* convert quad to double */
635                         info->res.type = fex_double;
636                         info->res.val.d = (double) ( dscl * ( dscl * info->op1.v
637                         break;
638                 }
640                 if (info->res.type == fex_nodata)
641                         /* couldn't do it */
642                         return;
643         }
645 stuff:
646         /* stick the result in the destination */
647         if (opf & 0x80) { /* conversion */
648                 if (opf & 0x10) { /* result is an int */
649                         switch (info->res.type) {
650                         case fex_llong:
651                                 info->res.val.i = (int) info->res.val.l;
652                                 break;
654                         case fex_float:
655                                 info->res.val.i = (int) info->res.val.f;
656                                 break;
658                         case fex_double:
659                                 info->res.val.i = (int) info->res.val.d;
660                                 break;
662                         case fex_ldouble:
663                                 info->res.val.i = (int) info->res.val.q;
```



```
664                                 break;
666                         default:
667                                 break;
668 #endif /* ! codereview */
669                         }
670                         *(int*)FPreg(rd) = info->res.val.i;
671                         return;
672                 }
674                 switch (opf & 0xc) {
675                 case 0: /* result is long long */
676                         switch (info->res.type) {
677                         case fex_int:
678                                 info->res.val.l = (long long) info->res.val.i;
679                                 break;
681                         case fex_float:
682                                 info->res.val.l = (long long) info->res.val.f;
683                                 break;
685                         case fex_double:
686                                 info->res.val.l = (long long) info->res.val.d;
687                                 break;
689                         case fex_ldouble:
690                                 info->res.val.l = (long long) info->res.val.q;
691                                 break;
693                         default:
694                                 break;
695 #endif /* ! codereview */
696                         }
697                         *(long long*)FPREG(rd) = info->res.val.l;
698                         break;
700                 case 0x4: /* result is float */
701                         switch (info->res.type) {
702                         case fex_int:
703                                 info->res.val.f = (float) info->res.val.i;
704                                 break;
706                         case fex_llong:
707                                 info->res.val.f = (float) info->res.val.l;
708                                 break;
710                         case fex_double:
711                                 info->res.val.f = (float) info->res.val.d;
712                                 break;
714                         case fex_ldouble:
715                                 info->res.val.f = (float) info->res.val.q;
716                                 break;
718                         default:
719                                 break;
720 #endif /* ! codereview */
721                         }
722                         *(float*)FPreg(rd) = info->res.val.f;
723                         break;
725                 case 0x8: /* result is double */
726                         switch (info->res.type) {
727                         case fex_int:
728                                 info->res.val.d = (double) info->res.val.i;
729                                 break;
```

```
731                         case fex_llong:
732                                 info->res.val.d = (double) info->res.val.l;
733                                 break;
734
735                         case fex_float:
736                                 info->res.val.d = (double) info->res.val.f;
737                                 break;
738
739                         case fex_ldouble:
740                                 info->res.val.d = (double) info->res.val.q;
741                                 break;
742
743                         default:
744                                 break;
745 #endif /* ! codereview */
746                         }
747                         *(double*)FPREG(rd) = info->res.val.d;
748                         break;
749
750                 case 0xc: /* result is long double */
751                         switch (info->res.type) {
752                         case fex_int:
753                                 info->res.val.q = (long double) info->res.val.i;
754                                 break;
755
756                         case fex_llong:
757                                 info->res.val.q = (long double) info->res.val.l;
758                                 break;
759
760                         case fex_float:
761                                 info->res.val.q = (long double) info->res.val.f;
762                                 break;
763
764                         case fex_double:
765                                 info->res.val.q = (long double) info->res.val.d;
766                                 break;
767
768                         default:
769                                 break;
770 #endif /* ! codereview */
771                         }
772                         *(long double*)FPREG(rd) = info->res.val.q;
773                         break;
774                 }
775                 return;
776         }
777
778         if ((opf & 0xf0) == 0x60) { /* fsmuld, fdmulq */
779                 switch (opf & 0xc0) {
780                 case 0x8: /* result is double */
781                         switch (info->res.type) {
782                         case fex_int:
783                                 info->res.val.d = (double) info->res.val.i;
784                                 break;
785
786                         case fex_llong:
787                                 info->res.val.d = (double) info->res.val.l;
788                                 break;
789
790                         case fex_float:
791                                 info->res.val.d = (double) info->res.val.f;
792                                 break;
793
794                         case fex_ldouble:
795                                 info->res.val.d = (double) info->res.val.q;
```



```
796                                 break;
797
798                         default:
799                                 break;
800 #endif /* ! codereview */
801                         }
802                         *(double*)FPREG(rd) = info->res.val.d;
803                         break;
804
805                 case 0xc: /* result is long double */
806                         switch (info->res.type) {
807                         case fex_int:
808                                 info->res.val.q = (long double) info->res.val.i;
809                                 break;
810
811                         case fex_llong:
812                                 info->res.val.q = (long double) info->res.val.l;
813                                 break;
814
815                         case fex_float:
816                                 info->res.val.q = (long double) info->res.val.f;
817                                 break;
818
819                         case fex_double:
820                                 info->res.val.q = (long double) info->res.val.d;
821                                 break;
822
823                         default:
824                                 break;
825 #endif /* ! codereview */
826                         }
827                         *(long double*)FPREG(rd) = info->res.val.q;
828                         break;
829                 }
830                 return;
831         }
832
833         switch (opf & 3) { /* other arithmetic op */
834         case 1: /* result is float */
835                 switch (info->res.type) {
836                 case fex_int:
837                         info->res.val.f = (float) info->res.val.i;
838                         break;
839
840                 case fex_llong:
841                         info->res.val.f = (float) info->res.val.l;
842                         break;
843
844                 case fex_double:
845                         info->res.val.f = (float) info->res.val.d;
846                         break;
847
848                 case fex_ldouble:
849                         info->res.val.f = (float) info->res.val.q;
850                         break;
851
852                 default:
853                         break;
854 #endif /* ! codereview */
855                 }
856                 *(float*)FPreg(rd) = info->res.val.f;
857                 break;
858
859         case 2: /* result is double */
860                 switch (info->res.type) {
861                 case fex_int:
```

```
862                         info->res.val.d = (double) info->res.val.i;
863                         break;

865                 case fex_llong:
866                         info->res.val.d = (double) info->res.val.l;
867                         break;

869                 case fex_float:
870                         info->res.val.d = (double) info->res.val.f;
871                         break;

873                 case fex_ldouble:
874                         info->res.val.d = (double) info->res.val.q;
875                         break;

877                 default:
878                         break;
879 #endif /* ! codereview */
880                 }
881                 *(double*)FPREG(rd) = info->res.val.d;
882                 break;

884         case 3: /* result is long double */
885                 switch (info->res.type) {
886                 case fex_int:
887                         info->res.val.q = (long double) info->res.val.i;
888                         break;

890                 case fex_llong:
891                         info->res.val.q = (long double) info->res.val.l;
892                         break;

894                 case fex_float:
895                         info->res.val.q = (long double) info->res.val.f;
896                         break;

898                 case fex_double:
899                         info->res.val.q = (long double) info->res.val.d;
900                         break;

902                 default:
903 #endif /* ! codereview */
904                         break;
905                 }
906                 *(long double*)FPREG(rd) = info->res.val.q;
907                 break;
908         }
909 }
910 #endif  /* defined(__sparc) */
```

```
**********************************************************
   39094 Sun May 11 12:16:10 2014
new/usr/src/lib/libm/common/m9x/__fex_sse.c
**********************************************************
______unchanged_portion_omitted_

 338 /*
 339  * Inspect a scalar SSE instruction that incurred an invalid operation
 340  * exception to determine which type of exception it was.
 341  */
 342 static enum fex_exception
 343 __fex_get_sse_invalid_type(sseinst_t *inst)
 344 {
 345         enum fp_class_type      t1, t2;

 347         /* check op2 for signaling nan */
 348         t2 = ((int)inst->op & DOUBLE)? my_fp_class(&inst->op2->d[0]) :
 349             my_fp_classf(&inst->op2->f[0]);
 350         if (t2 == fp_signaling)
 351                 return fex_inv_snan;

 353         /* eliminate all single-operand instructions */
 354         switch (inst->op) {
 355         case cvtsd2ss:
 356         case cvtss2sd:
 357                 /* hmm, this shouldn't have happened */
 358                 return (enum fex_exception) -1;

 360         case sqrtss:
 361         case sqrtsd:
 362                 return fex_inv_sqrt;

 364         case cvtss2si:
 365         case cvtsd2si:
 366         case cvttss2si:
 367         case cvttsd2si:
 368         case cvtss2siq:
 369         case cvtsd2siq:
 370         case cvttss2siq:
 371         case cvttsd2siq:
 372                 return fex_inv_int;
 373         default:
 374                 break;
 375 #endif /* ! codereview */
 376         }

 378         /* check op1 for signaling nan */
 379         t1 = ((int)inst->op & DOUBLE)? my_fp_class(&inst->op1->d[0]) :
 380             my_fp_classf(&inst->op1->f[0]);
 381         if (t1 == fp_signaling)
 382                 return fex_inv_snan;

 384         /* check two-operand instructions for other cases */
 385         switch (inst->op) {
 386         case cmpss:
 387         case cmpsd:
 388         case minss:
 389         case minsd:
 390         case maxss:
 391         case maxsd:
 392         case comiss:
 393         case comisd:
 394                 return fex_inv_cmp;

 396         case addss:
 397         case addsd:
```



```
 398         case subss:
 399         case subsd:
 400                 if (t1 == fp_infinity && t2 == fp_infinity)
 401                         return fex_inv_isi;
 402                 break;

 404         case mulss:
 405         case mulsd:
 406                 if ((t1 == fp_zero && t2 == fp_infinity) ||
 407                     (t2 == fp_zero && t1 == fp_infinity))
 408                         return fex_inv_zmi;
 409                 break;

 411         case divss:
 412         case divsd:
 413                 if (t1 == fp_zero && t2 == fp_zero)
 414                         return fex_inv_zdz;
 415                 if (t1 == fp_infinity && t2 == fp_infinity)
 416                         return fex_inv_idi;
 417         default:
 418                 break;
 419 #endif /* ! codereview */
 420         }

 422         return (enum fex_exception)-1;
 423 }

 425 /* inline templates */
 426 extern void sse_cmpeqss(float *, float *, int *);
 427 extern void sse_cmpltss(float *, float *, int *);
 428 extern void sse_cmpless(float *, float *, int *);
 429 extern void sse_cmpunordss(float *, float *, int *);
 430 extern void sse_minss(float *, float *, float *);
 431 extern void sse_maxss(float *, float *, float *);
 432 extern void sse_addss(float *, float *, float *);
 433 extern void sse_subss(float *, float *, float *);
 434 extern void sse_mulss(float *, float *, float *);
 435 extern void sse_divss(float *, float *, float *);
 436 extern void sse_sqrtss(float *, float *);
 437 extern void sse_ucomiss(float *, float *);
 438 extern void sse_comiss(float *, float *);
 439 extern void sse_cvtss2sd(float *, double *);
 440 extern void sse_cvtsi2ss(int *, float *);
 441 extern void sse_cvttss2si(float *, int *);
 442 extern void sse_cvtss2si(float *, int *);
 443 #ifdef __amd64
 444 extern void sse_cvtsi2ssq(long long *, float *);
 445 extern void sse_cvttss2siq(float *, long long *);
 446 extern void sse_cvtss2siq(float *, long long *);
 447 #endif
 448 extern void sse_cmpeqsd(double *, double *, long long *);
 449 extern void sse_cmpltsd(double *, double *, long long *);
 450 extern void sse_cmplesd(double *, double *, long long *);
 451 extern void sse_cmpunordsd(double *, double *, long long *);
 452 extern void sse_minsd(double *, double *, double *);
 453 extern void sse_maxsd(double *, double *, double *);
 454 extern void sse_addsd(double *, double *, double *);
 455 extern void sse_subsd(double *, double *, double *);
 456 extern void sse_mulsd(double *, double *, double *);
 457 extern void sse_divsd(double *, double *, double *);
 458 extern void sse_sqrtsd(double *, double *);
 459 extern void sse_ucomisd(double *, double *);
 460 extern void sse_comisd(double *, double *);
 461 extern void sse_cvtsd2ss(double *, float *);
 462 extern void sse_cvtsi2sd(int *, double *);
 463 extern void sse_cvttsd2si(double *, int *);
```

```
464 extern void sse_cvtsd2si(double *, int *);
465 #ifdef __amd64
466 extern void sse_cvtsi2sdq(long long *, double *);
467 extern void sse_cvttsd2siq(double *, long long *);
468 extern void sse_cvtsd2siq(double *, long long *);
469 #endif

471 /*
472  * Fill in *info with the operands, default untrapped result, and
473  * flags produced by a scalar SSE instruction, and return the type
474  * of trapped exception (if any).  On entry, the mxcsr must have
475  * all exceptions masked and all flags clear.  The same conditions
476  * will hold on exit.
477  *
478  * This routine does not work if the instruction specified by *inst
479  * is not a scalar instruction.
480  */
481 enum fex_exception
482 __fex_get_sse_op(ucontext_t *uap, sseinst_t *inst, fex_info_t *info)
483 {
484         unsigned int    e, te, mxcsr, oldmxcsr, subnorm;

486         /*
487          * Perform the operation with traps disabled and check the
488          * exception flags.  If the underflow trap was enabled, also
489          * check for an exact subnormal result.
490          */
491         __fenv_getmxcsr(&oldmxcsr);
492         subnorm = 0;
493         if ((int)inst->op & DOUBLE) {
494                 if (inst->op == cvtsi2sd) {
495                         info->op1.type = fex_int;
496                         info->op1.val.i = inst->op2->i[0];
497                         info->op2.type = fex_nodata;
498                 } else if (inst->op == cvtsi2sdq) {
499                         info->op1.type = fex_llong;
500                         info->op1.val.l = inst->op2->l[0];
501                         info->op2.type = fex_nodata;
502                 } else if (inst->op == sqrtsd || inst->op == cvtsd2ss ||
503                     inst->op == cvttsd2si || inst->op == cvtsd2si ||
504                     inst->op == cvttsd2siq || inst->op == cvtsd2siq) {
505                         info->op1.type = fex_double;
506                         info->op1.val.d = inst->op2->d[0];
507                         info->op2.type = fex_nodata;
508                 } else {
509                         info->op1.type = fex_double;
510                         info->op1.val.d = inst->op1->d[0];
511                         info->op2.type = fex_double;
512                         info->op2.val.d = inst->op2->d[0];
513                 }
514                 info->res.type = fex_double;
515                 switch (inst->op) {
516                 case cmpsd:
517                         info->op = fex_cmp;
518                         info->res.type = fex_llong;
519                         switch (inst->imm & 3) {
520                         case 0:
521                                 sse_cmpeqsd(&info->op1.val.d, &info->op2.val.d,
522                                     &info->res.val.l);
523                                 break;

525                         case 1:
526                                 sse_cmpltsd(&info->op1.val.d, &info->op2.val.d,
527                                     &info->res.val.l);
528                                 break;
```



```
530                         case 2:
531                                 sse_cmplesd(&info->op1.val.d, &info->op2.val.d,
532                                     &info->res.val.l);
533                                 break;

535                         case 3:
536                                 sse_cmpunordsd(&info->op1.val.d,
537                                     &info->op2.val.d, &info->res.val.l);
538                         }
539                         if (inst->imm & 4)
540                                 info->res.val.l ^= 0xffffffffffffffffull;
541                         break;

543                 case minsd:
544                         info->op = fex_other;
545                         sse_minsd(&info->op1.val.d, &info->op2.val.d,
546                             &info->res.val.d);
547                         break;

549                 case maxsd:
550                         info->op = fex_other;
551                         sse_maxsd(&info->op1.val.d, &info->op2.val.d,
552                             &info->res.val.d);
553                         break;

555                 case addsd:
556                         info->op = fex_add;
557                         sse_addsd(&info->op1.val.d, &info->op2.val.d,
558                             &info->res.val.d);
559                         if (my_fp_class(&info->res.val.d) == fp_subnormal)
560                                 subnorm = 1;
561                         break;

563                 case subsd:
564                         info->op = fex_sub;
565                         sse_subsd(&info->op1.val.d, &info->op2.val.d,
566                             &info->res.val.d);
567                         if (my_fp_class(&info->res.val.d) == fp_subnormal)
568                                 subnorm = 1;
569                         break;

571                 case mulsd:
572                         info->op = fex_mul;
573                         sse_mulsd(&info->op1.val.d, &info->op2.val.d,
574                             &info->res.val.d);
575                         if (my_fp_class(&info->res.val.d) == fp_subnormal)
576                                 subnorm = 1;
577                         break;

579                 case divsd:
580                         info->op = fex_div;
581                         sse_divsd(&info->op1.val.d, &info->op2.val.d,
582                             &info->res.val.d);
583                         if (my_fp_class(&info->res.val.d) == fp_subnormal)
584                                 subnorm = 1;
585                         break;

587                 case sqrtsd:
588                         info->op = fex_sqrt;
589                         sse_sqrtsd(&info->op1.val.d, &info->res.val.d);
590                         break;

592                 case cvtsd2ss:
593                         info->op = fex_cnvt;
594                         info->res.type = fex_float;
595                         sse_cvtsd2ss(&info->op1.val.d, &info->res.val.f);
```

```
596                         if (my_fp_classf(&info->res.val.f) == fp_subnormal)
597                                 subnorm = 1;
598                         break;

600                 case cvtsi2sd:
601                         info->op = fex_cnvt;
602                         sse_cvtsi2sd(&info->op1.val.i, &info->res.val.d);
603                         break;

605                 case cvttsd2si:
606                         info->op = fex_cnvt;
607                         info->res.type = fex_int;
608                         sse_cvttsd2si(&info->op1.val.d, &info->res.val.i);
609                         break;

611                 case cvtsd2si:
612                         info->op = fex_cnvt;
613                         info->res.type = fex_int;
614                         sse_cvtsd2si(&info->op1.val.d, &info->res.val.i);
615                         break;

617 #ifdef __amd64
618                 case cvtsi2sdq:
619                         info->op = fex_cnvt;
620                         sse_cvtsi2sdq(&info->op1.val.l, &info->res.val.d);
621                         break;

623                 case cvttsd2siq:
624                         info->op = fex_cnvt;
625                         info->res.type = fex_llong;
626                         sse_cvttsd2siq(&info->op1.val.d, &info->res.val.l);
627                         break;

629                 case cvtsd2siq:
630                         info->op = fex_cnvt;
631                         info->res.type = fex_llong;
632                         sse_cvtsd2siq(&info->op1.val.d, &info->res.val.l);
633                         break;
634 #endif

636                 case ucomisd:
637                         info->op = fex_cmp;
638                         info->res.type = fex_nodata;
639                         sse_ucomisd(&info->op1.val.d, &info->op2.val.d);
640                         break;

642                 case comisd:
643                         info->op = fex_cmp;
644                         info->res.type = fex_nodata;
645                         sse_comisd(&info->op1.val.d, &info->op2.val.d);
646                         break;
647                 default:
648                         break;
649 #endif /* ! codereview */
650                 }
651         } else {
652                 if (inst->op == cvtsi2ss) {
653                         info->op1.type = fex_int;
654                         info->op1.val.i = inst->op2->i[0];
655                         info->op2.type = fex_nodata;
656                 } else if (inst->op == cvtsi2ssq) {
657                         info->op1.type = fex_llong;
658                         info->op1.val.l = inst->op2->l[0];
659                         info->op2.type = fex_nodata;
660                 } else if (inst->op == sqrtss || inst->op == cvtss2sd ||
661                     inst->op == cvttss2si || inst->op == cvtss2si ||
```



```
662                     inst->op == cvttss2siq || inst->op == cvtss2siq) {
663                         info->op1.type = fex_float;
664                         info->op1.val.f = inst->op2->f[0];
665                         info->op2.type = fex_nodata;
666                 } else {
667                         info->op1.type = fex_float;
668                         info->op1.val.f = inst->op1->f[0];
669                         info->op2.type = fex_float;
670                         info->op2.val.f = inst->op2->f[0];
671                 }
672                 info->res.type = fex_float;
673                 switch (inst->op) {
674                 case cmpss:
675                         info->op = fex_cmp;
676                         info->res.type = fex_int;
677                         switch (inst->imm & 3) {
678                         case 0:
679                                 sse_cmpeqss(&info->op1.val.f, &info->op2.val.f,
680                                     &info->res.val.i);
681                                 break;

683                         case 1:
684                                 sse_cmpltss(&info->op1.val.f, &info->op2.val.f,
685                                     &info->res.val.i);
686                                 break;

688                         case 2:
689                                 sse_cmpless(&info->op1.val.f, &info->op2.val.f,
690                                     &info->res.val.i);
691                                 break;

693                         case 3:
694                                 sse_cmpunordss(&info->op1.val.f,
695                                     &info->op2.val.f, &info->res.val.i);
696                         }
697                         if (inst->imm & 4)
698                                 info->res.val.i ^= 0xffffffffu;
699                         break;

701                 case minss:
702                         info->op = fex_other;
703                         sse_minss(&info->op1.val.f, &info->op2.val.f,
704                             &info->res.val.f);
705                         break;

707                 case maxss:
708                         info->op = fex_other;
709                         sse_maxss(&info->op1.val.f, &info->op2.val.f,
710                             &info->res.val.f);
711                         break;

713                 case addss:
714                         info->op = fex_add;
715                         sse_addss(&info->op1.val.f, &info->op2.val.f,
716                             &info->res.val.f);
717                         if (my_fp_classf(&info->res.val.f) == fp_subnormal)
718                                 subnorm = 1;
719                         break;

721                 case subss:
722                         info->op = fex_sub;
723                         sse_subss(&info->op1.val.f, &info->op2.val.f,
724                             &info->res.val.f);
725                         if (my_fp_classf(&info->res.val.f) == fp_subnormal)
726                                 subnorm = 1;
727                         break;
```

```
729                 case mulss:
730                         info->op = fex_mul;
731                         sse_mulss(&info->op1.val.f, &info->op2.val.f,
732                             &info->res.val.f);
733                         if (my_fp_classf(&info->res.val.f) == fp_subnormal)
734                                 subnorm = 1;
735                         break;
736
737                 case divss:
738                         info->op = fex_div;
739                         sse_divss(&info->op1.val.f, &info->op2.val.f,
740                             &info->res.val.f);
741                         if (my_fp_classf(&info->res.val.f) == fp_subnormal)
742                                 subnorm = 1;
743                         break;
744
745                 case sqrtss:
746                         info->op = fex_sqrt;
747                         sse_sqrtss(&info->op1.val.f, &info->res.val.f);
748                         break;
749
750                 case cvtss2sd:
751                         info->op = fex_cnvt;
752                         info->res.type = fex_double;
753                         sse_cvtss2sd(&info->op1.val.f, &info->res.val.d);
754                         break;
755
756                 case cvtsi2ss:
757                         info->op = fex_cnvt;
758                         sse_cvtsi2ss(&info->op1.val.i, &info->res.val.f);
759                         break;
760
761                 case cvttss2si:
762                         info->op = fex_cnvt;
763                         info->res.type = fex_int;
764                         sse_cvttss2si(&info->op1.val.f, &info->res.val.i);
765                         break;
766
767                 case cvtss2si:
768                         info->op = fex_cnvt;
769                         info->res.type = fex_int;
770                         sse_cvtss2si(&info->op1.val.f, &info->res.val.i);
771                         break;
772
773 #ifdef __amd64
774                 case cvtsi2ssq:
775                         info->op = fex_cnvt;
776                         sse_cvtsi2ssq(&info->op1.val.l, &info->res.val.f);
777                         break;
778
779                 case cvttss2siq:
780                         info->op = fex_cnvt;
781                         info->res.type = fex_llong;
782                         sse_cvttss2siq(&info->op1.val.f, &info->res.val.l);
783                         break;
784
785                 case cvtss2siq:
786                         info->op = fex_cnvt;
787                         info->res.type = fex_llong;
788                         sse_cvtss2siq(&info->op1.val.f, &info->res.val.l);
789                         break;
790 #endif
791
792                 case ucomiss:
793                         info->op = fex_cmp;
```



```
794                         info->res.type = fex_nodata;
795                         sse_ucomiss(&info->op1.val.f, &info->op2.val.f);
796                         break;
797
798                 case comiss:
799                         info->op = fex_cmp;
800                         info->res.type = fex_nodata;
801                         sse_comiss(&info->op1.val.f, &info->op2.val.f);
802                         break;
803                 default:
804                         break;
805 #endif /* ! codereview */
806                 }
807         }
808         __fenv_getmxcsr(&mxcsr);
809         info->flags = mxcsr & 0x3d;
810         __fenv_setmxcsr(&oldmxcsr);
811
812         /* determine which exception would have been trapped */
813         te = ~(uap->uc_mcontext.fpregs.fp_reg_set.fpchip_state.mxcsr
814             >> 7) & 0x3d;
815         e = mxcsr & te;
816         if (e & FE_INVALID)
817                 return __fex_get_sse_invalid_type(inst);
818         if (e & FE_DIVBYZERO)
819                 return fex_division;
820         if (e & FE_OVERFLOW)
821                 return fex_overflow;
822         if ((e & FE_UNDERFLOW) || (subnorm && (te & FE_UNDERFLOW)))
823                 return fex_underflow;
824         if (e & FE_INEXACT)
825                 return fex_inexact;
826         return (enum fex_exception)-1;
827 }
828
829 /*
830  * Emulate a SIMD SSE instruction to determine which exceptions occur
831  * in each part.  For i = 0, 1, 2, and 3, set e[i] to indicate the
832  * trapped exception that would occur if the i-th part of the SIMD
833  * instruction were executed in isolation; set e[i] to -1 if no
834  * trapped exception would occur in this part.  Also fill in info[i]
835  * with the corresponding operands, default untrapped result, and
836  * flags.
837  *
838  * This routine does not work if the instruction specified by *inst
839  * is not a SIMD instruction.
840  */
841 void
842 __fex_get_simd_op(ucontext_t *uap, sseinst_t *inst, enum fex_exception *e,
843     fex_info_t *info)
844 {
845         sseinst_t       dummy;
846         int             i;
847
848         e[0] = e[1] = e[2] = e[3] = -1;
849
850         /* perform each part of the SIMD operation */
851         switch (inst->op) {
852         case cmpps:
853                 dummy.op = cmpss;
854                 dummy.imm = inst->imm;
855                 for (i = 0; i < 4; i++) {
856                         dummy.op1 = (sseoperand_t *)&inst->op1->f[i];
857                         dummy.op2 = (sseoperand_t *)&inst->op2->f[i];
858                         e[i] = __fex_get_sse_op(uap, &dummy, &info[i]);
859                 }
```

```
860                 break;

862         case minps:
863                 dummy.op = minss;
864                 for (i = 0; i < 4; i++) {
865                         dummy.op1 = (sseoperand_t *)&inst->op1->f[i];
866                         dummy.op2 = (sseoperand_t *)&inst->op2->f[i];
867                         e[i] = __fex_get_sse_op(uap, &dummy, &info[i]);
868                 }
869                 break;

871         case maxps:
872                 dummy.op = maxss;
873                 for (i = 0; i < 4; i++) {
874                         dummy.op1 = (sseoperand_t *)&inst->op1->f[i];
875                         dummy.op2 = (sseoperand_t *)&inst->op2->f[i];
876                         e[i] = __fex_get_sse_op(uap, &dummy, &info[i]);
877                 }
878                 break;

880         case addps:
881                 dummy.op = addss;
882                 for (i = 0; i < 4; i++) {
883                         dummy.op1 = (sseoperand_t *)&inst->op1->f[i];
884                         dummy.op2 = (sseoperand_t *)&inst->op2->f[i];
885                         e[i] = __fex_get_sse_op(uap, &dummy, &info[i]);
886                 }
887                 break;

889         case subps:
890                 dummy.op = subss;
891                 for (i = 0; i < 4; i++) {
892                         dummy.op1 = (sseoperand_t *)&inst->op1->f[i];
893                         dummy.op2 = (sseoperand_t *)&inst->op2->f[i];
894                         e[i] = __fex_get_sse_op(uap, &dummy, &info[i]);
895                 }
896                 break;

898         case mulps:
899                 dummy.op = mulss;
900                 for (i = 0; i < 4; i++) {
901                         dummy.op1 = (sseoperand_t *)&inst->op1->f[i];
902                         dummy.op2 = (sseoperand_t *)&inst->op2->f[i];
903                         e[i] = __fex_get_sse_op(uap, &dummy, &info[i]);
904                 }
905                 break;

907         case divps:
908                 dummy.op = divss;
909                 for (i = 0; i < 4; i++) {
910                         dummy.op1 = (sseoperand_t *)&inst->op1->f[i];
911                         dummy.op2 = (sseoperand_t *)&inst->op2->f[i];
912                         e[i] = __fex_get_sse_op(uap, &dummy, &info[i]);
913                 }
914                 break;

916         case sqrtps:
917                 dummy.op = sqrtss;
918                 for (i = 0; i < 4; i++) {
919                         dummy.op1 = (sseoperand_t *)&inst->op1->f[i];
920                         dummy.op2 = (sseoperand_t *)&inst->op2->f[i];
921                         e[i] = __fex_get_sse_op(uap, &dummy, &info[i]);
922                 }
923                 break;

925         case cvtdq2ps:
```



```
926                 dummy.op = cvtsi2ss;
927                 for (i = 0; i < 4; i++) {
928                         dummy.op1 = (sseoperand_t *)&inst->op1->f[i];
929                         dummy.op2 = (sseoperand_t *)&inst->op2->i[i];
930                         e[i] = __fex_get_sse_op(uap, &dummy, &info[i]);
931                 }
932                 break;

934         case cvttps2dq:
935                 dummy.op = cvttss2si;
936                 for (i = 0; i < 4; i++) {
937                         dummy.op1 = (sseoperand_t *)&inst->op1->i[i];
938                         dummy.op2 = (sseoperand_t *)&inst->op2->f[i];
939                         e[i] = __fex_get_sse_op(uap, &dummy, &info[i]);
940                 }
941                 break;

943         case cvtps2dq:
944                 dummy.op = cvtss2si;
945                 for (i = 0; i < 4; i++) {
946                         dummy.op1 = (sseoperand_t *)&inst->op1->i[i];
947                         dummy.op2 = (sseoperand_t *)&inst->op2->f[i];
948                         e[i] = __fex_get_sse_op(uap, &dummy, &info[i]);
949                 }
950                 break;

952         case cvtpi2ps:
953                 dummy.op = cvtsi2ss;
954                 for (i = 0; i < 2; i++) {
955                         dummy.op1 = (sseoperand_t *)&inst->op1->f[i];
956                         dummy.op2 = (sseoperand_t *)&inst->op2->i[i];
957                         e[i] = __fex_get_sse_op(uap, &dummy, &info[i]);
958                 }
959                 break;

961         case cvttps2pi:
962                 dummy.op = cvttss2si;
963                 for (i = 0; i < 2; i++) {
964                         dummy.op1 = (sseoperand_t *)&inst->op1->i[i];
965                         dummy.op2 = (sseoperand_t *)&inst->op2->f[i];
966                         e[i] = __fex_get_sse_op(uap, &dummy, &info[i]);
967                 }
968                 break;

970         case cvtps2pi:
971                 dummy.op = cvtss2si;
972                 for (i = 0; i < 2; i++) {
973                         dummy.op1 = (sseoperand_t *)&inst->op1->i[i];
974                         dummy.op2 = (sseoperand_t *)&inst->op2->f[i];
975                         e[i] = __fex_get_sse_op(uap, &dummy, &info[i]);
976                 }
977                 break;

979         case cmppd:
980                 dummy.op = cmpsd;
981                 dummy.imm = inst->imm;
982                 for (i = 0; i < 2; i++) {
983                         dummy.op1 = (sseoperand_t *)&inst->op1->d[i];
984                         dummy.op2 = (sseoperand_t *)&inst->op2->d[i];
985                         e[i] = __fex_get_sse_op(uap, &dummy, &info[i]);
986                 }
987                 break;

989         case minpd:
990                 dummy.op = minsd;
991                 for (i = 0; i < 2; i++) {
```

```
                        dummy.op1 = (sseoperand_t *)&inst->op1->d[i];
                        dummy.op2 = (sseoperand_t *)&inst->op2->d[i];
                        e[i] = __fex_get_sse_op(uap, &dummy, &info[i]);
                }
                break;

        case maxpd:
                dummy.op = maxsd;
                for (i = 0; i < 2; i++) {
                        dummy.op1 = (sseoperand_t *)&inst->op1->d[i];
                        dummy.op2 = (sseoperand_t *)&inst->op2->d[i];
                        e[i] = __fex_get_sse_op(uap, &dummy, &info[i]);
                }
                break;

        case addpd:
                dummy.op = addsd;
                for (i = 0; i < 2; i++) {
                        dummy.op1 = (sseoperand_t *)&inst->op1->d[i];
                        dummy.op2 = (sseoperand_t *)&inst->op2->d[i];
                        e[i] = __fex_get_sse_op(uap, &dummy, &info[i]);
                }
                break;

        case subpd:
                dummy.op = subsd;
                for (i = 0; i < 2; i++) {
                        dummy.op1 = (sseoperand_t *)&inst->op1->d[i];
                        dummy.op2 = (sseoperand_t *)&inst->op2->d[i];
                        e[i] = __fex_get_sse_op(uap, &dummy, &info[i]);
                }
                break;

        case mulpd:
                dummy.op = mulsd;
                for (i = 0; i < 2; i++) {
                        dummy.op1 = (sseoperand_t *)&inst->op1->d[i];
                        dummy.op2 = (sseoperand_t *)&inst->op2->d[i];
                        e[i] = __fex_get_sse_op(uap, &dummy, &info[i]);
                }
                break;

        case divpd:
                dummy.op = divsd;
                for (i = 0; i < 2; i++) {
                        dummy.op1 = (sseoperand_t *)&inst->op1->d[i];
                        dummy.op2 = (sseoperand_t *)&inst->op2->d[i];
                        e[i] = __fex_get_sse_op(uap, &dummy, &info[i]);
                }
                break;

        case sqrtpd:
                dummy.op = sqrtsd;
                for (i = 0; i < 2; i++) {
                        dummy.op1 = (sseoperand_t *)&inst->op1->d[i];
                        dummy.op2 = (sseoperand_t *)&inst->op2->d[i];
                        e[i] = __fex_get_sse_op(uap, &dummy, &info[i]);
                }
                break;

        case cvtpi2pd:
        case cvtdq2pd:
                dummy.op = cvtsi2sd;
                for (i = 0; i < 2; i++) {
                        dummy.op1 = (sseoperand_t *)&inst->op1->d[i];
                        dummy.op2 = (sseoperand_t *)&inst->op2->i[i];
```



```
                        e[i] = __fex_get_sse_op(uap, &dummy, &info[i]);
                }
                break;

        case cvttpd2pi:
        case cvttpd2dq:
                dummy.op = cvttsd2si;
                for (i = 0; i < 2; i++) {
                        dummy.op1 = (sseoperand_t *)&inst->op1->i[i];
                        dummy.op2 = (sseoperand_t *)&inst->op2->d[i];
                        e[i] = __fex_get_sse_op(uap, &dummy, &info[i]);
                }
                break;

        case cvtpd2pi:
        case cvtpd2dq:
                dummy.op = cvtsd2si;
                for (i = 0; i < 2; i++) {
                        dummy.op1 = (sseoperand_t *)&inst->op1->i[i];
                        dummy.op2 = (sseoperand_t *)&inst->op2->d[i];
                        e[i] = __fex_get_sse_op(uap, &dummy, &info[i]);
                }
                break;

        case cvtps2pd:
                dummy.op = cvtss2sd;
                for (i = 0; i < 2; i++) {
                        dummy.op1 = (sseoperand_t *)&inst->op1->d[i];
                        dummy.op2 = (sseoperand_t *)&inst->op2->f[i];
                        e[i] = __fex_get_sse_op(uap, &dummy, &info[i]);
                }
                break;

        case cvtpd2ps:
                dummy.op = cvtsd2ss;
                for (i = 0; i < 2; i++) {
                        dummy.op1 = (sseoperand_t *)&inst->op1->f[i];
                        dummy.op2 = (sseoperand_t *)&inst->op2->d[i];
                        e[i] = __fex_get_sse_op(uap, &dummy, &info[i]);
                }
        default:
                break;
#endif /* ! codereview */
        }
}

/*
 * Store the result value from *info in the destination of the scalar
 * SSE instruction specified by *inst.  If no result is given but the
 * exception is underflow or overflow, supply the default trapped result.
 *
 * This routine does not work if the instruction specified by *inst
 * is not a scalar instruction.
 */
void
__fex_st_sse_result(ucontext_t *uap, sseinst_t *inst, enum fex_exception e,
    fex_info_t *info)
{
        int             i = 0;
        long long       l = 0L;;
        float           f = 0.0, fscl;
        double          d = 0.0L, dscl;
        int             i;
        long long       l;
        float           f, fscl;
        double          d, dscl;
```

```
1121         /* for compares that write eflags, just set the flags
1122            to indicate "unordered" */
1123         if (inst->op == ucomiss || inst->op == comiss ||
1124             inst->op == ucomisd || inst->op == comisd) {
1125                 uap->uc_mcontext.gregs[REG_PS] |= 0x45;
1126                 return;
1127         }

1129         /* if info doesn't specify a result value, try to generate
1130            the default trapped result */
1131         if (info->res.type == fex_nodata) {
1132                 /* set scale factors for exponent wrapping */
1133                 switch (e) {
1134                 case fex_overflow:
1135                         fscl = 1.262177448e-29f; /* 2^-96 */
1136                         dscl = 6.441148769597133308e-232; /* 2^-768 */
1137                         break;

1139                 case fex_underflow:
1140                         fscl = 7.922816251e+28f; /* 2^96 */
1141                         dscl = 1.5525180923007089350e+231; /* 2^768 */
1142                         break;

1144                 default:
1145                         (void) __fex_get_sse_op(uap, inst, info);
1146                         if (info->res.type == fex_nodata)
1147                                 return;
1148                         goto stuff;
1149                 }

1151                 /* generate the wrapped result */
1152                 if (inst->op == cvtsd2ss) {
1153                         info->op1.type = fex_double;
1154                         info->op1.val.d = inst->op2->d[0];
1155                         info->op2.type = fex_nodata;
1156                         info->res.type = fex_float;
1157                         info->res.val.f = (float)(fscl * (fscl *
1158                             info->op1.val.d));
1159                 } else if ((int)inst->op & DOUBLE) {
1160                         info->op1.type = fex_double;
1161                         info->op1.val.d = inst->op1->d[0];
1162                         info->op2.type = fex_double;
1163                         info->op2.val.d = inst->op2->d[0];
1164                         info->res.type = fex_double;
1165                         switch (inst->op) {
1166                         case addsd:
1167                                 info->res.val.d = dscl * (dscl *
1168                                     info->op1.val.d + dscl * info->op2.val.d);
1169                                 break;

1171                         case subsd:
1172                                 info->res.val.d = dscl * (dscl *
1173                                     info->op1.val.d - dscl * info->op2.val.d);
1174                                 break;

1176                         case mulsd:
1177                                 info->res.val.d = (dscl * info->op1.val.d) *
1178                                     (dscl * info->op2.val.d);
1179                                 break;

1181                         case divsd:
1182                                 info->res.val.d = (dscl * info->op1.val.d) /
1183                                     (info->op2.val.d / dscl);
1184                                 break;
```



```
1186                         default:
1187                                 return;
1188                         }
1189                 } else {
1190                         info->op1.type = fex_float;
1191                         info->op1.val.f = inst->op1->f[0];
1192                         info->op2.type = fex_float;
1193                         info->op2.val.f = inst->op2->f[0];
1194                         info->res.type = fex_float;
1195                         switch (inst->op) {
1196                         case addss:
1197                                 info->res.val.f = fscl * (fscl *
1198                                     info->op1.val.f + fscl * info->op2.val.f);
1199                                 break;

1201                         case subss:
1202                                 info->res.val.f = fscl * (fscl *
1203                                     info->op1.val.f - fscl * info->op2.val.f);
1204                                 break;

1206                         case mulss:
1207                                 info->res.val.f = (fscl * info->op1.val.f) *
1208                                     (fscl * info->op2.val.f);
1209                                 break;

1211                         case divss:
1212                                 info->res.val.f = (fscl * info->op1.val.f) /
1213                                     (info->op2.val.f / fscl);
1214                                 break;

1216                         default:
1217                                 return;
1218                         }
1219                 }
1220         }

1222         /* put the result in the destination */
1223 stuff:
1224         if (inst->op == cmpss || inst->op == cvttss2si || inst->op == cvtss2si
1225             || inst->op == cvttsd2si || inst->op == cvtsd2si) {
1226                 switch (info->res.type) {
1227                 case fex_int:
1228                         i = info->res.val.i;
1229                         break;

1231                 case fex_llong:
1232                         i = info->res.val.l;
1233                         break;

1235                 case fex_float:
1236                         i = info->res.val.f;
1237                         break;

1239                 case fex_double:
1240                         i = info->res.val.d;
1241                         break;

1243                 case fex_ldouble:
1244                         i = info->res.val.q;
1245                         break;

1247                 default:
1248                         break;
1249 #endif /* ! codereview */
1250                 }
1251                 inst->op1->i[0] = i;
```

```
1252         } else if (inst->op == cmpsd || inst->op == cvttss2siq ||
1253             inst->op == cvtss2siq || inst->op == cvttsd2siq ||
1254             inst->op == cvtsd2siq) {
1255                 switch (info->res.type) {
1256                 case fex_int:
1257                         l = info->res.val.i;
1258                         break;

1260                 case fex_llong:
1261                         l = info->res.val.l;
1262                         break;

1264                 case fex_float:
1265                         l = info->res.val.f;
1266                         break;

1268                 case fex_double:
1269                         l = info->res.val.d;
1270                         break;

1272                 case fex_ldouble:
1273                         l = info->res.val.q;
1274                         break;

1276                 default:
1277                         break;
1278 #endif /* ! codereview */
1279                 }
1280                 inst->op1->l[0] = l;
1281         } else if ((((int)inst->op & DOUBLE) && inst->op != cvtsd2ss) ||
1282             inst->op == cvtss2sd) {
1283                 switch (info->res.type) {
1284                 case fex_int:
1285                         d = info->res.val.i;
1286                         break;

1288                 case fex_llong:
1289                         d = info->res.val.l;
1290                         break;

1292                 case fex_float:
1293                         d = info->res.val.f;
1294                         break;

1296                 case fex_double:
1297                         d = info->res.val.d;
1298                         break;

1300                 case fex_ldouble:
1301                         d = info->res.val.q;
1302                         break;

1304                 default:
1305                         break;
1306 #endif /* ! codereview */
1307                 }
1308                 inst->op1->d[0] = d;
1309         } else {
1310                 switch (info->res.type) {
1311                 case fex_int:
1312                         f = info->res.val.i;
1313                         break;

1315                 case fex_llong:
1316                         f = info->res.val.l;
1317                         break;
```



```
1319                 case fex_float:
1320                         f = info->res.val.f;
1321                         break;

1323                 case fex_double:
1324                         f = info->res.val.d;
1325                         break;

1327                 case fex_ldouble:
1328                         f = info->res.val.q;
1329                         break;

1331                 default:
1332                         break;
1333 #endif /* ! codereview */
1334                 }
1335                 inst->op1->f[0] = f;
1336         }
1337 }

1339 /*
1340  * Store the results from a SIMD instruction.  For each i, store
1341  * the result value from info[i] in the i-th part of the destination
1342  * of the SIMD SSE instruction specified by *inst.  If no result
1343  * is given but the exception indicated by e[i] is underflow or
1344  * overflow, supply the default trapped result.
1345  *
1346  * This routine does not work if the instruction specified by *inst
1347  * is not a SIMD instruction.
1348  */
1349 void
1350 __fex_st_simd_result(ucontext_t *uap, sseinst_t *inst, enum fex_exception *e,
1351     fex_info_t *info)
1352 {
1353         sseinst_t       dummy;
1354         int             i;

1356         /* store each part */
1357         switch (inst->op) {
1358         case cmpps:
1359                 dummy.op = cmpss;
1360                 dummy.imm = inst->imm;
1361                 for (i = 0; i < 4; i++) {
1362                         dummy.op1 = (sseoperand_t *)&inst->op1->f[i];
1363                         dummy.op2 = (sseoperand_t *)&inst->op2->f[i];
1364                         __fex_st_sse_result(uap, &dummy, e[i], &info[i]);
1365                 }
1366                 break;

1368         case minps:
1369                 dummy.op = minss;
1370                 for (i = 0; i < 4; i++) {
1371                         dummy.op1 = (sseoperand_t *)&inst->op1->f[i];
1372                         dummy.op2 = (sseoperand_t *)&inst->op2->f[i];
1373                         __fex_st_sse_result(uap, &dummy, e[i], &info[i]);
1374                 }
1375                 break;

1377         case maxps:
1378                 dummy.op = maxss;
1379                 for (i = 0; i < 4; i++) {
1380                         dummy.op1 = (sseoperand_t *)&inst->op1->f[i];
1381                         dummy.op2 = (sseoperand_t *)&inst->op2->f[i];
1382                         __fex_st_sse_result(uap, &dummy, e[i], &info[i]);
1383                 }
```

```
1384                    break;
1385
1386            case addps:
1387                    dummy.op = addss;
1388                    for (i = 0; i < 4; i++) {
1389                            dummy.op1 = (sseoperand_t *)&inst->op1->f[i];
1390                            dummy.op2 = (sseoperand_t *)&inst->op2->f[i];
1391                            __fex_st_sse_result(uap, &dummy, e[i], &info[i]);
1392                    }
1393                    break;
1394
1395            case subps:
1396                    dummy.op = subss;
1397                    for (i = 0; i < 4; i++) {
1398                            dummy.op1 = (sseoperand_t *)&inst->op1->f[i];
1399                            dummy.op2 = (sseoperand_t *)&inst->op2->f[i];
1400                            __fex_st_sse_result(uap, &dummy, e[i], &info[i]);
1401                    }
1402                    break;
1403
1404            case mulps:
1405                    dummy.op = mulss;
1406                    for (i = 0; i < 4; i++) {
1407                            dummy.op1 = (sseoperand_t *)&inst->op1->f[i];
1408                            dummy.op2 = (sseoperand_t *)&inst->op2->f[i];
1409                            __fex_st_sse_result(uap, &dummy, e[i], &info[i]);
1410                    }
1411                    break;
1412
1413            case divps:
1414                    dummy.op = divss;
1415                    for (i = 0; i < 4; i++) {
1416                            dummy.op1 = (sseoperand_t *)&inst->op1->f[i];
1417                            dummy.op2 = (sseoperand_t *)&inst->op2->f[i];
1418                            __fex_st_sse_result(uap, &dummy, e[i], &info[i]);
1419                    }
1420                    break;
1421
1422            case sqrtps:
1423                    dummy.op = sqrtss;
1424                    for (i = 0; i < 4; i++) {
1425                            dummy.op1 = (sseoperand_t *)&inst->op1->f[i];
1426                            dummy.op2 = (sseoperand_t *)&inst->op2->f[i];
1427                            __fex_st_sse_result(uap, &dummy, e[i], &info[i]);
1428                    }
1429                    break;
1430
1431            case cvtdq2ps:
1432                    dummy.op = cvtsi2ss;
1433                    for (i = 0; i < 4; i++) {
1434                            dummy.op1 = (sseoperand_t *)&inst->op1->f[i];
1435                            dummy.op2 = (sseoperand_t *)&inst->op2->i[i];
1436                            __fex_st_sse_result(uap, &dummy, e[i], &info[i]);
1437                    }
1438                    break;
1439
1440            case cvttps2dq:
1441                    dummy.op = cvttss2si;
1442                    for (i = 0; i < 4; i++) {
1443                            dummy.op1 = (sseoperand_t *)&inst->op1->i[i];
1444                            dummy.op2 = (sseoperand_t *)&inst->op2->f[i];
1445                            __fex_st_sse_result(uap, &dummy, e[i], &info[i]);
1446                    }
1447                    break;
1448
1449            case cvtps2dq:
```



```
1450                    dummy.op = cvtss2si;
1451                    for (i = 0; i < 4; i++) {
1452                            dummy.op1 = (sseoperand_t *)&inst->op1->i[i];
1453                            dummy.op2 = (sseoperand_t *)&inst->op2->f[i];
1454                            __fex_st_sse_result(uap, &dummy, e[i], &info[i]);
1455                    }
1456                    break;
1457
1458            case cvtpi2ps:
1459                    dummy.op = cvtsi2ss;
1460                    for (i = 0; i < 2; i++) {
1461                            dummy.op1 = (sseoperand_t *)&inst->op1->f[i];
1462                            dummy.op2 = (sseoperand_t *)&inst->op2->i[i];
1463                            __fex_st_sse_result(uap, &dummy, e[i], &info[i]);
1464                    }
1465                    break;
1466
1467            case cvttps2pi:
1468                    dummy.op = cvttss2si;
1469                    for (i = 0; i < 2; i++) {
1470                            dummy.op1 = (sseoperand_t *)&inst->op1->i[i];
1471                            dummy.op2 = (sseoperand_t *)&inst->op2->f[i];
1472                            __fex_st_sse_result(uap, &dummy, e[i], &info[i]);
1473                    }
1474                    break;
1475
1476            case cvtps2pi:
1477                    dummy.op = cvtss2si;
1478                    for (i = 0; i < 2; i++) {
1479                            dummy.op1 = (sseoperand_t *)&inst->op1->i[i];
1480                            dummy.op2 = (sseoperand_t *)&inst->op2->f[i];
1481                            __fex_st_sse_result(uap, &dummy, e[i], &info[i]);
1482                    }
1483                    break;
1484
1485            case cmppd:
1486                    dummy.op = cmpsd;
1487                    dummy.imm = inst->imm;
1488                    for (i = 0; i < 2; i++) {
1489                            dummy.op1 = (sseoperand_t *)&inst->op1->d[i];
1490                            dummy.op2 = (sseoperand_t *)&inst->op2->d[i];
1491                            __fex_st_sse_result(uap, &dummy, e[i], &info[i]);
1492                    }
1493                    break;
1494
1495            case minpd:
1496                    dummy.op = minsd;
1497                    for (i = 0; i < 2; i++) {
1498                            dummy.op1 = (sseoperand_t *)&inst->op1->d[i];
1499                            dummy.op2 = (sseoperand_t *)&inst->op2->d[i];
1500                            __fex_st_sse_result(uap, &dummy, e[i], &info[i]);
1501                    }
1502                    break;
1503
1504            case maxpd:
1505                    dummy.op = maxsd;
1506                    for (i = 0; i < 2; i++) {
1507                            dummy.op1 = (sseoperand_t *)&inst->op1->d[i];
1508                            dummy.op2 = (sseoperand_t *)&inst->op2->d[i];
1509                            __fex_st_sse_result(uap, &dummy, e[i], &info[i]);
1510                    }
1511                    break;
1512
1513            case addpd:
1514                    dummy.op = addsd;
1515                    for (i = 0; i < 2; i++) {
```

```
1516                        dummy.op1 = (sseoperand_t *)&inst->op1->d[i];
1517                        dummy.op2 = (sseoperand_t *)&inst->op2->d[i];
1518                        __fex_st_sse_result(uap, &dummy, e[i], &info[i]);
1519                }
1520                break;

1522        case subpd:
1523                dummy.op = subsd;
1524                for (i = 0; i < 2; i++) {
1525                        dummy.op1 = (sseoperand_t *)&inst->op1->d[i];
1526                        dummy.op2 = (sseoperand_t *)&inst->op2->d[i];
1527                        __fex_st_sse_result(uap, &dummy, e[i], &info[i]);
1528                }
1529                break;

1531        case mulpd:
1532                dummy.op = mulsd;
1533                for (i = 0; i < 2; i++) {
1534                        dummy.op1 = (sseoperand_t *)&inst->op1->d[i];
1535                        dummy.op2 = (sseoperand_t *)&inst->op2->d[i];
1536                        __fex_st_sse_result(uap, &dummy, e[i], &info[i]);
1537                }
1538                break;

1540        case divpd:
1541                dummy.op = divsd;
1542                for (i = 0; i < 2; i++) {
1543                        dummy.op1 = (sseoperand_t *)&inst->op1->d[i];
1544                        dummy.op2 = (sseoperand_t *)&inst->op2->d[i];
1545                        __fex_st_sse_result(uap, &dummy, e[i], &info[i]);
1546                }
1547                break;

1549        case sqrtpd:
1550                dummy.op = sqrtsd;
1551                for (i = 0; i < 2; i++) {
1552                        dummy.op1 = (sseoperand_t *)&inst->op1->d[i];
1553                        dummy.op2 = (sseoperand_t *)&inst->op2->d[i];
1554                        __fex_st_sse_result(uap, &dummy, e[i], &info[i]);
1555                }
1556                break;

1558        case cvtpi2pd:
1559        case cvtdq2pd:
1560                dummy.op = cvtsi2sd;
1561                for (i = 0; i < 2; i++) {
1562                        dummy.op1 = (sseoperand_t *)&inst->op1->d[i];
1563                        dummy.op2 = (sseoperand_t *)&inst->op2->i[i];
1564                        __fex_st_sse_result(uap, &dummy, e[i], &info[i]);
1565                }
1566                break;

1568        case cvttpd2pi:
1569        case cvttpd2dq:
1570                dummy.op = cvttsd2si;
1571                for (i = 0; i < 2; i++) {
1572                        dummy.op1 = (sseoperand_t *)&inst->op1->i[i];
1573                        dummy.op2 = (sseoperand_t *)&inst->op2->d[i];
1574                        __fex_st_sse_result(uap, &dummy, e[i], &info[i]);
1575                }
1576                /* for cvttpd2dq, zero the high 64 bits of the destination */
1577                if (inst->op == cvttpd2dq)
1578                        inst->op1->l[1] = 0ll;
1579                break;

1581        case cvtpd2pi:
```



```
1582        case cvtpd2dq:
1583                dummy.op = cvtsd2si;
1584                for (i = 0; i < 2; i++) {
1585                        dummy.op1 = (sseoperand_t *)&inst->op1->i[i];
1586                        dummy.op2 = (sseoperand_t *)&inst->op2->d[i];
1587                        __fex_st_sse_result(uap, &dummy, e[i], &info[i]);
1588                }
1589                /* for cvtpd2dq, zero the high 64 bits of the destination */
1590                if (inst->op == cvtpd2dq)
1591                        inst->op1->l[1] = 0ll;
1592                break;

1594        case cvtps2pd:
1595                dummy.op = cvtss2sd;
1596                for (i = 0; i < 2; i++) {
1597                        dummy.op1 = (sseoperand_t *)&inst->op1->d[i];
1598                        dummy.op2 = (sseoperand_t *)&inst->op2->f[i];
1599                        __fex_st_sse_result(uap, &dummy, e[i], &info[i]);
1600                }
1601                break;

1603        case cvtpd2ps:
1604                dummy.op = cvtsd2ss;
1605                for (i = 0; i < 2; i++) {
1606                        dummy.op1 = (sseoperand_t *)&inst->op1->f[i];
1607                        dummy.op2 = (sseoperand_t *)&inst->op2->d[i];
1608                        __fex_st_sse_result(uap, &dummy, e[i], &info[i]);
1609                }
1610                /* zero the high 64 bits of the destination */
1611                inst->op1->l[1] = 0ll;

1613        default:
1614                break;
1615 #endif /* ! codereview */
1616        }
1617 }

1619 #endif /* ! codereview */
```

****
9376 Sun May 11 12:16:12 2014
new/usr/src/lib/libm/common/m9x/fex_log.c
****
_____unchanged_portion_omitted_

```
108 #ifdef __sparcv9
109 #define FRAMEP(X)       (struct frame *)((char*)(X)+(((long)(X)&1)?2047:0))
110 #else
111 #define FRAMEP(X)       (struct frame *)(X)
112 #endif

114 #ifdef _LP64
115 #define PDIG            "16"
116 #else
117 #define PDIG            "8"
118 #endif

120 /* look for a matching exc_list; return 1 if one is found,
121    otherwise add this one to the list and return 0 */
122 static int check_exc_list(char *addr, unsigned long code, char *stk,
123     struct frame *fp)
124 {
125         struct exc_list *l, *ll = NULL;
125         struct exc_list *l, *ll;
126         struct frame    *f;
127         int             i, n;

129         if (list) {
130                 for (l = list; l; ll = l, l = l->next) {
131                         if (l->addr != addr || l->code != code)
132                                 continue;
133                         if (log_depth < 1 || l->nstack < 1)
134                                 return 1;
135                         if (l->stack[0] != stk)
136                                 continue;
137                         n = 1;
138                         for (i = 1, f = fp; i < log_depth && i < l->nstack &&
139                             f && f->fr_savpc; i++, f = FRAMEP(f->fr_savfp))
140                                 if (l->stack[i] != (char *)f->fr_savpc) {
141                                         n = 0;
142                                         break;
143                                 }
144                         if (n)
145                                 return 1;
146                 }
147         }

149         /* create a new exc_list structure and tack it on the list */
150         for (n = 1, f = fp; n < log_depth && f && f->fr_savpc;
151             n++, f = FRAMEP(f->fr_savfp)) ;
152         if ((l = (struct exc_list *)malloc(sizeof(struct exc_list) +
153             (n - 1) * sizeof(char *))) != NULL) {
154                 l->next = NULL;
155                 l->addr = addr;
156                 l->code = code;
157                 l->nstack = ((log_depth < 1)? 0 : n);
158                 l->stack[0] = stk;
159                 for (i = 1; i < n; i++) {
160                         l->stack[i] = (char *)fp->fr_savpc;
161                         fp = FRAMEP(fp->fr_savfp);
162                 }
163                 if (list)
164                         ll->next = l;
165                 else
166                         list = l;
```



```
167         }
168         return 0;
169 }
```

_____unchanged_portion_omitted_

****
11316 Sun May 11 12:16:13 2014
new/usr/src/lib/libm/common/m9x/fma.c
****
_____unchanged_portion_omitted_

```
 56 #define half    C[0].d
 57 #define two     C[1].d
 58 #define two52   C[2].d
 59 #define two27   C[3].d
 60 #define twom26  C[4].d
 61 #define twom32  C[5].d
 62 #define twom64  C[6].d
 63 #define huge    C[7].d
 64 #define tiny    C[8].d
 65 #define tiny2   C[9].d

 67 static const unsigned int fsr_rm = 0xc0000000u;

 69 /*
 70  * fma for SPARC: 64-bit double precision, big-endian
 71  */
 72 double
 73 __fma(double x, double y, double z) {
 74         union {
 75                 unsigned i[2];
 76                 double d;
 77         } xx, yy, zz;
 78         double xhi, yhi, xlo, ylo, t;
 79         unsigned int xy0, xy1, xy2, xy3, z0, z1, z2, z3, fsr, rm, sticky;
 79         unsigned int xy0, xy1, xy2, xy3, z0, z1, z2, z3, rm, sticky;
 80         unsigned int fsr;
 80         int hx, hy, hz, ex, ey, ez, exy, sxy, sz, e, ibit;
 81         volatile double dummy;

 83         /* extract the high order words of the arguments */
 84         xx.d = x;
 85         yy.d = y;
 86         zz.d = z;
 87         hx = xx.i[0] & ~0x80000000;
 88         hy = yy.i[0] & ~0x80000000;
 89         hz = zz.i[0] & ~0x80000000;

 91         /* dispense with inf, nan, and zero cases */
 92         if (hx >= 0x7ff00000 || hy >= 0x7ff00000 || (hx | xx.i[1]) == 0 ||
 93                 (hy | yy.i[1]) == 0)    /* x or y is inf, nan, or zero */
 94                 return (x * y + z);

 96         if (hz >= 0x7ff00000)   /* z is inf or nan */
 97                 return (x + z); /* avoid spurious under/overflow in x * y */

 99         if ((hz | zz.i[1]) == 0)        /* z is zero */
100                 /*
101                  * x * y isn't zero but could underflow to zero,
102                  * so don't add z, lest we perturb the sign
103                  */
104                 return (x * y);

106         /*
107          * now x, y, and z are all finite and nonzero; save the fsr and
108          * set round-to-negative-infinity mode (and clear nonstandard
109          * mode before we try to scale subnormal operands)
110          */
111         __fenv_getfsr32(&fsr);
112         __fenv_setfsr32(&fsr_rm);
```



```
114         /* extract signs and exponents, and normalize subnormals */
115         sxy = (xx.i[0] ^ yy.i[0]) & 0x80000000;
116         sz = zz.i[0] & 0x80000000;
117         ex = hx >> 20;
118         if (!ex) {
119                 xx.d = x * two52;
120                 ex = ((xx.i[0] & ~0x80000000) >> 20) - 52;
121         }
122         ey = hy >> 20;
123         if (!ey) {
124                 yy.d = y * two52;
125                 ey = ((yy.i[0] & ~0x80000000) >> 20) - 52;
126         }
127         ez = hz >> 20;
128         if (!ez) {
129                 zz.d = z * two52;
130                 ez = ((zz.i[0] & ~0x80000000) >> 20) - 52;
131         }

133         /* multiply x*y to 106 bits */
134         exy = ex + ey - 0x3ff;
135         xx.i[0] = (xx.i[0] & 0xfffff) | 0x3ff00000;
136         yy.i[0] = (yy.i[0] & 0xfffff) | 0x3ff00000;
137         x = xx.d;
138         y = yy.d;
139         xhi = ((x + twom26) + two27) - two27;
140         yhi = ((y + twom26) + two27) - two27;
141         xlo = x - xhi;
142         ylo = y - yhi;
143         x *= y;
144         y = ((xhi * yhi - x) + xhi * ylo + xlo * yhi) + xlo * ylo;
145         if (x >= two) {
146                 x *= half;
147                 y *= half;
148                 exy++;
149         }

151         /* extract the significands */
152         xx.d = x;
153         xy0 = (xx.i[0] & 0xfffff) | 0x100000;
154         xy1 = xx.i[1];
155         yy.d = t = y + twom32;
156         xy2 = yy.i[1];
157         yy.d = (y - (t - twom32)) + twom64;
158         xy3 = yy.i[1];
159         z0 = (zz.i[0] & 0xfffff) | 0x100000;
160         z1 = zz.i[1];
161         z2 = z3 = 0;

163         /*
164          * now x*y is represented by sxy, exy, and xy[0-3], and z is
165          * represented likewise; swap if need be so |xy| <= |z|
166          */
167         if (exy > ez || (exy == ez && (xy0 > z0 || (xy0 == z0 &&
168                 (xy1 > z1 || (xy1 == z1 && (xy2 | xy3) != 0)))))) {
169                 e = sxy; sxy = sz; sz = e;
170                 e = exy; exy = ez; ez = e;
171                 e = xy0; xy0 = z0; z0 = e;
172                 e = xy1; xy1 = z1; z1 = e;
173                 z2 = xy2; xy2 = 0;
174                 z3 = xy3; xy3 = 0;
175         }

177         /* shift the significand of xy keeping a sticky bit */
178         e = ez - exy;
179         if (e > 116) {
```

```
180                 xy0 = xy1 = xy2 = 0;
181                 xy3 = 1;
182         } else if (e >= 96) {
183                 sticky = xy3 | xy2 | xy1 | ((xy0 << 1) << (127 - e));
184                 xy3 = xy0 >> (e - 96);
185                 if (sticky)
186                         xy3 |= 1;
187                 xy0 = xy1 = xy2 = 0;
188         } else if (e >= 64) {
189                 sticky = xy3 | xy2 | ((xy1 << 1) << (95 - e));
190                 xy3 = (xy1 >> (e - 64)) | ((xy0 << 1) << (95 - e));
191                 if (sticky)
192                         xy3 |= 1;
193                 xy2 = xy0 >> (e - 64);
194                 xy0 = xy1 = 0;
195         } else if (e >= 32) {
196                 sticky = xy3 | ((xy2 << 1) << (63 - e));
197                 xy3 = (xy2 >> (e - 32)) | ((xy1 << 1) << (63 - e));
198                 if (sticky)
199                         xy3 |= 1;
200                 xy2 = (xy1 >> (e - 32)) | ((xy0 << 1) << (63 - e));
201                 xy1 = xy0 >> (e - 32);
202                 xy0 = 0;
203         } else if (e) {
204                 sticky = (xy3 << 1) << (31 - e);
205                 xy3 = (xy3 >> e) | ((xy2 << 1) << (31 - e));
206                 if (sticky)
207                         xy3 |= 1;
208                 xy2 = (xy2 >> e) | ((xy1 << 1) << (31 - e));
209                 xy1 = (xy1 >> e) | ((xy0 << 1) << (31 - e));
210                 xy0 >>= e;
211         }

213         /* if this is a magnitude subtract, negate the significand of xy */
214         if (sxy ^ sz) {
215                 xy0 = ~xy0;
216                 xy1 = ~xy1;
217                 xy2 = ~xy2;
218                 xy3 = -xy3;
219                 if (xy3 == 0)
220                         if (++xy2 == 0)
221                                 if (++xy1 == 0)
222                                         xy0++;
223         }

225         /* add, propagating carries */
226         z3 += xy3;
227         e = (z3 < xy3);
228         z2 += xy2;
229         if (e) {
230                 z2++;
231                 e = (z2 <= xy2);
232         } else
233                 e = (z2 < xy2);
234         z1 += xy1;
235         if (e) {
236                 z1++;
237                 e = (z1 <= xy1);
238         } else
239                 e = (z1 < xy1);
240         z0 += xy0;
241         if (e)
242                 z0++;

244         /* postnormalize and collect rounding information into z2 */
245         if (ez < 1) {
```



```
246                 /* result is tiny; shift right until exponent is within range */
247                 e = 1 - ez;
248                 if (e > 56) {
249                         z2 = 1; /* result can't be exactly zero */
250                         z0 = z1 = 0;
251                 } else if (e >= 32) {
252                         sticky = z3 | z2 | ((z1 << 1) << (63 - e));
253                         z2 = (z1 >> (e - 32)) | ((z0 << 1) << (63 - e));
254                         if (sticky)
255                                 z2 |= 1;
256                         z1 = z0 >> (e - 32);
257                         z0 = 0;
258                 } else {
259                         sticky = z3 | (z2 << 1) << (31 - e);
260                         z2 = (z2 >> e) | ((z1 << 1) << (31 - e));
261                         if (sticky)
262                                 z2 |= 1;
263                         z1 = (z1 >> e) | ((z0 << 1) << (31 - e));
264                         z0 >>= e;
265                 }
266                 ez = 1;
267         } else if (z0 >= 0x200000) {
268                 /* carry out; shift right by one */
269                 sticky = (z2 & 1) | z3;
270                 z2 = (z2 >> 1) | (z1 << 31);
271                 if (sticky)
272                         z2 |= 1;
273                 z1 = (z1 >> 1) | (z0 << 31);
274                 z0 >>= 1;
275                 ez++;
276         } else {
277                 if (z0 < 0x100000 && (z0 | z1 | z2 | z3) != 0) {
278                         /*
279                          * borrow/cancellation; shift left as much as
280                          * exponent allows
281                          */
282                         while (!(z0 | (z1 & 0xffe00000)) && ez >= 33) {
283                                 z0 = z1;
284                                 z1 = z2;
285                                 z2 = z3;
286                                 z3 = 0;
287                                 ez -= 32;
288                         }
289                         while (z0 < 0x100000 && ez > 1) {
290                                 z0 = (z0 << 1) | (z1 >> 31);
291                                 z1 = (z1 << 1) | (z2 >> 31);
292                                 z2 = (z2 << 1) | (z3 >> 31);
293                                 z3 <<= 1;
294                                 ez--;
295                         }
296                 }
297                 if (z3)
298                         z2 |= 1;
299         }

301         /* get the rounding mode and clear current exceptions */
302         rm = fsr >> 30;
303         fsr &= ~FSR_CEXC;

305         /* strip off the integer bit, if there is one */
306         ibit = z0 & 0x100000;
307         if (ibit)
308                 z0 -= 0x100000;
309         else {
310                 ez = 0;
311                 if (!(z0 | z1 | z2)) { /* exact zero */
```

```
312                         zz.i[0] = rm == FSR_RM ? 0x80000000 : 0;
313                         zz.i[1] = 0;
314                         __fenv_setfsr32(&fsr);
315                         return (zz.d);
316                 }
317         }

319         /*
320          * flip the sense of directed roundings if the result is negative;
321          * the logic below applies to a positive result
322          */
323         if (sz)
324                 rm ^= rm >> 1;

326         /* round and raise exceptions */
327         if (z2) {
328                 fsr |= FSR_NXC;

330                 /* decide whether to round the fraction up */
331                 if (rm == FSR_RP || (rm == FSR_RN && (z2 > 0x80000000u ||
332                         (z2 == 0x80000000u && (z1 & 1))))) {
333                         /* round up and renormalize if necessary */
334                         if (++z1 == 0) {
335                                 if (++z0 == 0x100000) {
336                                         z0 = 0;
337                                         ez++;
338                                 }
339                         }
340                 }
341         }

343         /* check for under/overflow */
344         if (ez >= 0x7ff) {
345                 if (rm == FSR_RN || rm == FSR_RP) {
346                         zz.i[0] = sz | 0x7ff00000;
347                         zz.i[1] = 0;
348                 } else {
349                         zz.i[0] = sz | 0x7fefffff;
350                         zz.i[1] = 0xffffffff;
351                 }
352                 fsr |= FSR_OFC | FSR_NXC;
353         } else {
354                 zz.i[0] = sz | (ez << 20) | z0;
355                 zz.i[1] = z1;

357                 /*
358                  * !ibit => exact result was tiny before rounding,
359                  * z2 nonzero => result delivered is inexact
360                  */
361                 if (!ibit) {
362                         if (z2)
363                                 fsr |= FSR_UFC | FSR_NXC;
364                         else if (fsr & FSR_UFM)
365                                 fsr |= FSR_UFC;
366                 }
367         }

369         /* restore the fsr and emulate exceptions as needed */
370         if ((fsr & FSR_CEXC) & (fsr >> 23)) {
371                 __fenv_setfsr32(&fsr);
372                 if (fsr & FSR_OFC) {
373                         dummy = huge;
374                         dummy *= huge;
375                 } else if (fsr & FSR_UFC) {
376                         dummy = tiny;
377                         if (fsr & FSR_NXC)
```



```
378                                 dummy *= tiny;
379                         else
380                                 dummy -= tiny2;
381                 } else {
382                         dummy = huge;
383                         dummy += tiny;
384                 }
385         } else {
386                 fsr |= (fsr & 0x1f) << 5;
387                 __fenv_setfsr32(&fsr);
388         }
389         return (zz.d);
390 }
```

_____***unchanged_portion_omitted_***

****
5109 Sun May 11 12:16:15 2014
new/usr/src/lib/libm/common/m9x/nearbyint.c
****

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" 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)))))
                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))))) {
                if (rm == FE_UPWARD || (rm == FE_TONEAREST &&
                        (frac > 0x80000000u || (frac == 0x80000000) &&
                        (xx.i[LOWORD] & i)))) {
                        xx.i[LOWORD] += i;
                        if (xx.i[LOWORD] == 0)
                                xx.i[HIWORD]++;
                }
        }
        return (xx.d);
}
```

_____unchanged_portion_omitted_

```
**********************************************************
    4024 Sun May 11 12:16:17 2014
new/usr/src/lib/libm/common/m9x/nearbyintf.c
**********************************************************
```

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

#include "libm.h"
#include "fenv_synonyms.h"
#include <fenv.h>

float
__nearbyintf(float x) {
        union {
                unsigned i;
                float f;
        } xx;
        unsigned hx, sx, i, frac;
        int rm;

        xx.f = x;
        sx = xx.i & 0x80000000;
        hx = xx.i & ~0x80000000;

        /* handle trivial cases */
        if (hx >= 0x4b000000) { /* x is nan, inf, or already integral */
                if (hx > 0x7f800000)    /* x is nan */
                        return (x * x);         /* + -> * for Cheetah */
                return (x);
        } else if (hx == 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 < 0x3f800000) {
                if (rm == FE_UPWARD || (rm == FE_TONEAREST && hx > 0x3f000000))
                        xx.i = sx | 0x3f800000;
                else
                        xx.i = sx;
                return (xx.f);
        }

        /* round x at the integer bit */
        i = 1 << (0x96 - (hx >> 23));
        frac = hx & (i - 1);
        if (!frac)
                return (x);

        hx &= ~(i - 1);
        if (rm == FE_UPWARD || (rm == FE_TONEAREST && (frac > (i >> 1) ||
                ((frac == (i >> 1)) && (hx & i)))))
                (frac == (i >> 1)) && (hx & i))))
                xx.i = sx | (hx + i);
        else
                xx.i = sx | hx;
        return (xx.f);
}
```

_____unchanged_portion_omitted_

```
**********************************************************
    2430 Sun May 11 12:16:19 2014
new/usr/src/lib/libm/common/m9x/scalblnl.c
**********************************************************
______unchanged_portion_omitted_
```

****
68985 Sun May 11 12:16:20 2014
new/usr/src/lib/libm/common/m9x/tgamma.c
****
_____unchanged_portion_omitted_

```
1392 /* INDENT OFF */
1393 static const double
1394         /* 0.13486180573279076968979393577465291700 6 */
1395         t0z1   =  0.1348618057327907737708,
1396         t0z1_l = -4.08100777085782990225316-18,
1397         /* 0.46163214496836234126265954232572132846 8 */
1398         t0z2   =  0.4616321449683623567850,
1399         t0z2_l = -1.55223481628586768905216-17,
1400         /* 0.81977310110050060178786870492160699631 2 */
1401         t0z3   =  0.8197731011005006118708,
1402         t0z3_l = -1.00829451224871034983256-17;
1403         /* 1.13486180573279076968979393577465291700 6 */
1404 /* INDENT ON */

1406 /* gamma(x+i) for 0 <= x < 1  */
1407 static struct Double
1408 gam_n(int i, double x) {
1409         struct Double rr = {0.0L, 0.0L}, yy;
1409         struct Double rr, yy;
1410         double r1, r2, t2, z, xh, xl, yh, yl, zh, z1, z2, zl, x5, wh, wl;

1412         /* compute yy = gamma(x+1) */
1413         if (x > 0.2845) {
1414                 if (x > 0.6374) {
1415                         r1 = x - t0z3;
1416                         r2 = (double) ((float) (r1 - t0z3_l));
1417                         t2 = r1 - r2;
1418                         yy = GT3(r2, t2 - t0z3_l);
1419                 } else {
1420                         r1 = x - t0z2;
1421                         r2 = (double) ((float) (r1 - t0z2_l));
1422                         t2 = r1 - r2;
1423                         yy = GT2(r2, t2 - t0z2_l);
1424                 }
1425         } else {
1426                 r1 = x - t0z1;
1427                 r2 = (double) ((float) (r1 - t0z1_l));
1428                 t2 = r1 - r2;
1429                 yy = GT1(r2, t2 - t0z1_l);
1430         }

1432         /* compute gamma(x+i) = (x+i-1)*...*(x+1)*yy, 0<i<8 */
1433         switch (i) {
1434         case 0:         /* yy/x */
1435                 r1 = one / x;
1436                 xh = (double) ((float) x);      /* x is not tiny */
1437                 rr.h = (double) ((float) ((yy.h + yy.l) * r1));
1438                 rr.l = r1 * (yy.h - rr.h * xh) -
1439                         ((r1 * rr.h) * (x - xh) - r1 * yy.l);
1440                 break;
1441         case 1:         /* yy */
1442                 rr.h = yy.h;
1443                 rr.l = yy.l;
1444                 break;
1445         case 2:         /* (x+1)*yy */
1446                 z = x + one;    /* may not be exact */
1447                 zh = (double) ((float) z);
1448                 rr.h = zh * yy.h;
1449                 rr.l = z * yy.l + (x - (zh - one)) * yy.h;
1450                 break;
```



```
1451         case 3:         /* (x+2)*(x+1)*yy */
1452                 z1 = x + one;
1453                 z2 = x + 2.0;
1454                 z = z1 * z2;
1455                 xh = (double) ((float) z);
1456                 zh = (double) ((float) z1);
1457                 xl = (x - (zh - one)) * (z2 + zh) - (xh - zh * (zh + one));
1458                 rr.h = xh * yy.h;
1459                 rr.l = z * yy.l + xl * yy.h;
1460                 break;

1462         case 4:         /* (x+1)*(x+3)*(x+2)*yy */
1463                 z1 = x + 2.0;
1464                 z2 = (x + one) * (x + 3.0);
1465                 zh = z1;
1466                 __LO(zh) = 0;
1467                 __HI(zh) &= 0xffffffff8; /* zh 18 bits mantissa */
1468                 zl = x - (zh - 2.0);
1469                 z = z1 * z2;
1470                 xh = (double) ((float) z);
1471                 xl = zl * (z2 + zh * (z1 + zh)) - (xh - zh * (zh * zh - one));
1472                 rr.h = xh * yy.h;
1473                 rr.l = z * yy.l + xl * yy.h;
1474                 break;
1475         case 5:         /* ((x+1)*(x+4)*(x+2)*(x+3))*yy */
1476                 z1 = x + 2.0;
1477                 z2 = x + 3.0;
1478                 z = z1 * z2;
1479                 zh = (double) ((float) z1);
1480                 yh = (double) ((float) z);
1481                 yl = (x - (zh - 2.0)) * (z2 + zh) - (yh - zh * (zh + one));
1482                 z2 = z - 2.0;
1483                 z *= z2;
1484                 xh = (double) ((float) z);
1485                 xl = yl * (z2 + yh) - (xh - yh * (yh - 2.0));
1486                 rr.h = xh * yy.h;
1487                 rr.l = z * yy.l + xl * yy.h;
1488                 break;
1489         case 6:         /* ((x+1)*(x+2)*(x+3)*(x+4)*(x+5))*yy */
1490                 z1 = x + 2.0;
1491                 z2 = x + 3.0;
1492                 z = z1 * z2;
1493                 zh = (double) ((float) z1);
1494                 yh = (double) ((float) z);
1495                 z1 = x - (zh - 2.0);
1496                 yl = z1 * (z2 + zh) - (yh - zh * (zh + one));
1497                 z2 = z - 2.0;
1498                 x5 = x + 5.0;
1499                 z *= z2;
1500                 xh = (double) ((float) z);
1501                 zh += 3.0;
1502                 xl = yl * (z2 + yh) - (xh - yh * (yh - 2.0));
1503                                         /* xh+xl=(x+1)*...*(x+4) */
1504                 /* wh+wl=(x+5)*yy */
1505                 wh = (double) ((float) (x5 * (yy.h + yy.l)));
1506                 wl = (z1 * yy.h + x5 * yy.l) - (wh - zh * yy.h);
1507                 rr.h = wh * xh;
1508                 rr.l = z * wl + xl * wh;
1509                 break;
1510         case 7:         /* ((x+1)*(x+2)*(x+3)*(x+4)*(x+5)*(x+6))*yy */
1511                 z1 = x + 3.0;
1512                 z2 = x + 4.0;
1513                 z = z2 * z1;
1514                 zh = (double) ((float) z1);
1515                 yh = (double) ((float) z);      /* yh+yl = (x+3)(x+4) */
1516                 yl = (x - (zh - 3.0)) * (z2 + zh) - (yh - (zh * (zh + one)));
```

```
1517                 z1 = x + 6.0;
1518                 z2 = z - 2.0;   /* z2 = (x+2)*(x+5) */
1519                 z *= z2;
1520                 xh = (double) ((float) z);
1521                 xl = yl * (z2 + yh) - (xh - yh * (yh - 2.0));
1522                                         /* xh+xl=(x+2)*...*(x+5) */
1523                 /* wh+wl=(x+1)(x+6)*yy */
1524                 z2 -= 4.0;      /* z2 = (x+1)(x+6) */
1525                 wh = (double) ((float) (z2 * (yy.h + yy.l)));
1526                 wl = (z2 * yy.l + yl * yy.h) - (wh - (yh - 6.0) * yy.h);
1527                 rr.h = wh * xh;
1528                 rr.l = z * wl + xl * wh;
1529         }
1530         return (rr);
1531 }

1533 double
1534 tgamma(double x) {
1535         struct Double ss, ww;
1536         double t, t1, t2, t3, t4, t5, w, y, z, z1, z2, z3, z5;
1537         int i, j, k, m, ix, hx, xk;
1538         unsigned lx;

1540         hx = __HI(x);
1541         lx = __LO(x);
1542         ix = hx & 0x7fffffff;
1543         y = x;

1545         if (ix < 0x3ca00000)
1546                 return (one / x);       /* |x| < 2**-53 */
1547         if (ix >= 0x7ff00000)
1548                         /* +Inf -> +Inf, -Inf or NaN -> NaN */
1549                 return (x * ((hx < 0)? 0.0 : x));
1550         if (hx > 0x406573fa ||  /* x > 171.62... overflow to +inf */
1551             (hx == 0x406573fa && lx > 0xE561F647)) {
1552                 z = x / tiny;
1553                 return (z * z);
1554         }
1555         if (hx >= 0x40200000) { /* x >= 8 */
1556                 ww = large_gam(x, &m);
1557                 w = ww.h + ww.l;
1558                 __HI(w) += m << 20;
1559                 return (w);
1560         }
1561         if (hx > 0) {           /* 0 < x < 8 */
1561         if (hx > 0) {           /* x from 0 to 8 */
1562                 i = (int) x;
1563                 ww = gam_n(i, x - (double) i);
1564                 return (ww.h + ww.l);
1565         }

1567         /* negative x */
1568         /* INDENT OFF */
1569         /*
1570          * compute: xk =
1571          *      -2 ... x is an even int (-inf is even)
1572          *      -1 ... x is an odd int
1573          *      +0 ... x is not an int but chopped to an even int
1574          *      +1 ... x is not an int but chopped to an odd int
1575          */
1576         /* INDENT ON */
1577         xk = 0;
1578         if (ix >= 0x43300000) {
1579                 if (ix >= 0x43400000)
1580                         xk = -2;
1581                 else
```



```
1582                         xk = -2 + (lx & 1);
1583         } else if (ix >= 0x3ff00000) {
1584                 k = (ix >> 20) - 0x3ff;
1585                 if (k > 20) {
1586                         j = lx >> (52 - k);
1587                         if ((j << (52 - k)) == lx)
1588                                 xk = -2 + (j & 1);
1589                         else
1590                                 xk = j & 1;
1591                 } else {
1592                         j = ix >> (20 - k);
1593                         if ((j << (20 - k)) == ix && lx == 0)
1594                                 xk = -2 + (j & 1);
1595                         else
1596                                 xk = j & 1;
1597                 }
1598         }
1599         if (xk < 0)
1600                 /* ideally gamma(-n)= (-1)**(n+1) * inf, but c99 expect NaN */
1601                 return ((x - x) / (x - x));             /* 0/0 = NaN */


1604         /* negative underflow thresold */
1605         if (ix > 0x4066e000 || (ix == 0x4066e000 && lx > 11)) {
1606                 /* x < -183.0 - 11ulp */
1607                 z = tiny / x;
1608                 if (xk == 1)
1609                         z = -z;
1610                 return (z * tiny);
1611         }

1613         /* now compute gamma(x) by  -1/((sin(pi*y)/pi)*gamma(1+y)), y = -x */

1615         /*
1616          * First compute ss = -sin(pi*y)/pi , so that
1617          * gamma(x) = 1/(ss*gamma(1+y))
1618          */
1619         y = -x;
1620         j = (int) y;
1621         z = y - (double) j;
1622         if (z > 0.31830988618137906715377675)
1623                 if (z > 0.68169011381620932846223250)
1624                         ss = kpsin(one - z);
1625                 else
1626                         ss = kpcos(0.5 - z);
1627         else
1628                 ss = kpsin(z);
1629         if (xk == 0) {
1630                 ss.h = -ss.h;
1631                 ss.l = -ss.l;
1632         }

1634         /* Then compute ww = gamma(1+y), note that result scale to 2**m */
1635         m = 0;
1636         if (j < 7) {
1637                 ww = gam_n(j + 1, z);
1638         } else {
1639                 w = y + one;
1640                 if ((lx & 1) == 0) {    /* y+1 exact (note that y<184) */
1641                         ww = large_gam(w, &m);
1642                 } else {
1643                         t = w - one;
1644                         if (t == y) {   /* y+one exact */
1645                                 ww = large_gam(w, &m);
1646                         } else {        /* use y*gamma(y) */
1647                                 if (j == 7)
```

```
1648                                    ww = gam_n(j, z);
1649                            else
1650                                    ww = large_gam(y, &m);
1651                            t4 = ww.h + ww.l;
1652                            t1 = (double) ((float) y);
1653                            t2 = (double) ((float) t4);
1654                                            /* t4 will not be too large */
1655                            ww.l = y * (ww.l - (t2 - ww.h)) + (y - t1) * t2;
1656                            ww.h = t1 * t2;
1657                    }
1658            }
1659    }

1661    /* compute 1/(ss*ww) */
1662    t3 = ss.h + ss.l;
1663    t4 = ww.h + ww.l;
1664    t1 = (double) ((float) t3);
1665    t2 = (double) ((float) t4);
1666    z1 = ss.l - (t1 - ss.h);        /* (t1,z1) = ss */
1667    z2 = ww.l - (t2 - ww.h);        /* (t2,z2) = ww */
1668    t3 = t3 * t4;                   /* t3 = ss*ww */
1669    z3 = one / t3;                  /* z3 = 1/(ss*ww) */
1670    t5 = t1 * t2;
1671    z5 = z1 * t4 + t1 * z2;         /* (t5,z5) = ss*ww */
1672    t1 = (double) ((float) t3);     /* (t1,z1) = ss*ww */
1673    z1 = z5 - (t1 - t5);
1674    t2 = (double) ((float) z3);     /* leading 1/(ss*ww) */
1675    z2 = z3 * (t2 * z1 - (one - t2 * t1));
1676    z = t2 - z2;

1678    /* check whether z*2**-m underflow */
1679    if (m != 0) {
1680            hx = __HI(z);
1681            i = hx & 0x80000000;
1682            ix = hx ^ i;
1683            j = ix >> 20;
1684            if (j > m) {
1685                    ix -= m << 20;
1686                    __HI(z) = ix ^ i;
1687            } else if ((m - j) > 52) {
1688                    /* underflow */
1689                    if (xk == 0)
1690                            z = -tiny * tiny;
1691                    else
1692                            z = tiny * tiny;
1693            } else {
1694                    /* subnormal */
1695                    m -= 60;
1696                    t = one;
1697                    __HI(t) -= 60 << 20;
1698                    ix -= m << 20;
1699                    __HI(z) = ix ^ i;
1700                    z *= t;
1701            }
1702    }
1703    return (z);
1704 }
```

_____***unchanged_portion_omitted_***

****
15261 Sun May 11 12:16:22 2014
new/usr/src/lib/libm/common/m9x/tgammaf.c
****
_____***unchanged_portion_omitted_***

```
 390 /* INDENT OFF */
 391 static const double
 392 t0z1 = 0.134861805732790769689793935774652917006,
 393 t0z2 = 0.461632144968362341262659542325721328468,
 394 t0z3 = 0.819773101100500601787868704921606996312;
 395         /* 1.134861805732790769689793935774652917006 */
 396 /* INDENT ON */

 398 /*
 399  * gamma(x+i) for 0 <= x < 1
 400  */
 401 static double
 402 gam_n(int i, double x) {
 403         double rr = 0.0L, yy;
 403         double rr, yy;
 404         double z1, z2;

 406         /* compute yy = gamma(x+1) */
 407         if (x > 0.2845) {
 408                 if (x > 0.6374)
 409                         yy = GT3(x - t0z3);
 410                 else
 411                         yy = GT2(x - t0z2);
 412         } else
 413                 yy = GT1(x - t0z1);

 415         /* compute gamma(x+i) = (x+i-1)*...*(x+1)*yy, 0<i<8 */
 416         switch (i) {
 417         case 0:         /* yy/x */
 418                 rr = yy / x;
 419                 break;
 420         case 1:         /* yy */
 421                 rr = yy;
 422                 break;
 423         case 2:         /* (x+1)*yy */
 424                 rr = (x + one) * yy;
 425                 break;
 426         case 3:         /* (x+2)*(x+1)*yy */
 427                 rr = (x + one) * (x + two) * yy;
 428                 break;

 430         case 4:         /* (x+1)*(x+3)*(x+2)*yy */
 431                 rr = (x + one) * (x + two) * ((x + 3.0) * yy);
 432                 break;
 433         case 5:         /* ((x+1)*(x+4)*(x+2)*(x+3))*yy */
 434                 z1 = (x + two) * (x + 3.0) * yy;
 435                 z2 = (x + one) * (x + 4.0);
 436                 rr = z1 * z2;
 437                 break;
 438         case 6:         /* ((x+1)*(x+2)*(x+3)*(x+4)*(x+5))*yy */
 439                 z1 = (x + two) * (x + 3.0);
 440                 z2 = (x + 5.0) * yy;
 441                 rr = z1 * (z1 - two) * z2;
 442                 break;
 443         case 7:         /* ((x+1)*(x+2)*(x+3)*(x+4)*(x+5)*(x+6))*yy */
 444                 z1 = (x + two) * (x + 3.0);
 445                 z2 = (x + 5.0) * (x + 6.0) * yy;
 446                 rr = z1 * (z1 - two) * z2;
 447                 break;
 448         }
```



```
 449         return (rr);
 450 }

 452 float
 453 tgammaf(float xf) {
 454         float zf;
 455         double ss, ww;
 456         double x, y, z;
 457         int i, j, k, ix, hx, xk;

 459         hx = *(int *) &xf;
 460         ix = hx & 0x7fffffff;

 462         x = (double) xf;
 463         if (ix < 0x33800000)
 464                 return (1.0F / xf);     /* |x| < 2**-24 */

 466         if (ix >= 0x7f800000)
 467                 return (xf * ((hx < 0)? 0.0F : xf)); /* +-Inf or NaN */

 469         if (hx > 0x420C290F)    /* x > 35.040096283... overflow */
 470                 return (float)(x / tiny);

 472         if (hx >= 0x41000000)   /* x >= 8 */
 473                 return ((float) large_gam(x));

 475         if (hx > 0) {           /* 0 < x < 8 */
 475         if (hx > 0) {           /* x from 0 to 8 */
 476                 i = (int) xf;
 477                 return ((float) gam_n(i, x - (double) i));
 478         }

 480         /* negative x */
 481         /* INDENT OFF */
 482         /*
 483          * compute xk =
 484          *      -2 ... x is an even int (-inf is considered even)
 485          *      -1 ... x is an odd int
 486          *      +0 ... x is not an int but chopped to an even int
 487          *      +1 ... x is not an int but chopped to an odd int
 488          */
 489         /* INDENT ON */
 490         xk = 0;
 491         if (ix >= 0x4b000000) {
 492                 if (ix > 0x4b000000)
 493                         xk = -2;
 494                 else
 495                         xk = -2 + (ix & 1);
 496         } else if (ix >= 0x3f800000) {
 497                 k = (ix >> 23) - 0x7f;
 498                 j = ix >> (23 - k);
 499                 if ((j << (23 - k)) == ix)
 500                         xk = -2 + (j & 1);
 501                 else
 502                         xk = j & 1;
 503         }
 504         if (xk < 0) {
 505                 /* 0/0 invalid NaN, ideally gamma(-n)= (-1)**(n+1) * inf */
 506                 zf = xf - xf;
 507                 return (zf / zf);
 508         }

 510         /* negative underflow thresold */
 511         if (ix > 0x4224000B) {  /* x < -(41+11ulp) */
 512                 if (xk == 0)
 513                         z = -tiny;
```

```
514                 else
515                         z = tiny;
516                 return ((float)z);
517         }

519         /* INDENT OFF */
520         /* now compute gamma(x) by  -1/((sin(pi*y)/pi)*gamma(1+y)), y = -x */
521         /*
522          * First compute ss = -sin(pi*y)/pi , so that
523          * gamma(x) = 1/(ss*gamma(1+y))
524          */
525         /* INDENT ON */
526         y = -x;
527         j = (int) y;
528         z = y - (double) j;
529         if (z > 0.31830988618379067153776752)
530                 if (z > 0.68169011381620932846223254)
531                         ss = kpsin(one - z);
532                 else
533                         ss = kpcos(0.5 - z);
534         else
535                 ss = kpsin(z);
536         if (xk == 0)
537                 ss = -ss;

539         /* Then compute ww = gamma(1+y)  */
540         if (j < 7)
541                 ww = gam_n(j + 1, z);
542         else
543                 ww = large_gam(y + one);

545         /* return 1/(ss*ww) */
546         return ((float) (one / (ww * ss)));
547 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   40087 Sun May 11 12:16:24 2014
new/usr/src/lib/libm/common/m9x/tgammal.c
**********************************************************
______unchanged_portion_omitted_

/* INDENT OFF */
static const long double
        /* 0.134861805732790769689793935774652917006425111395524293982 33 */
#if defined(__x86)
t0z1   =  0.13486180573279076967793850549970358088 10L,
t0z1_l =  1.18554302749493361253927171502573796146 54e-20L,
#else
t0z1   =  0.13486180573279076968979393577465296 9168654L,
t0z1_l =  1.41020885886768794187391644861595146 74310e-37L,
#endif
        /* 0.461632144968362341262659542325721328468196204006446351295 99 */
#if defined(__x86)
t0z2   =  0.46163214496836234125381158432954720183 26L,
t0z2_l =  8.84795799617412663558532305039261747030 640e-21L,
#else
t0z2   =  0.46163214496836234126265954232572132343 318L,
t0z2_l =  5.03501162329616380465302666480916271611 101e-36L,
#endif
        /* 0.819773101100500601787868704921606996311744078462451791195 86 */
#if defined(__x86)
t0z3   =  0.81977310110050060178773362329351925836 817L,
t0z3_l =  1.35081628087737943565807705253457455625 6230e-22L
#else
t0z3   =  0.81977310110050060178786870492160695169 57449L,
t0z3_l =  4.46159991694701441904549261593355164885 7380e-35L
#endif
;
/* INDENT ON */

/*
 * gamma(x+i) for 0 <= x < 1
 */
static struct LDouble
gam_n(int i, long double x) {
        struct LDouble rr = {0.0L, 0.0L}, yy;
        struct LDouble rr, yy;
        long double r1, r2, t2, z, xh, xl, yh, yl, zh, z1, z2, zl, x5, wh, wl;

        /* compute yy = gamma(x+1) */
        if (x > 0.2845L) {
                if (x > 0.6374L) {
                        r1 = x - t0z3;
                        r2 = CHOPPED((r1 - t0z3_l));
                        t2 = r1 - r2;
                        yy = GT3(r2, t2 - t0z3_l);
                } else {
                        r1 = x - t0z2;
                        r2 = CHOPPED((r1 - t0z2_l));
                        t2 = r1 - r2;
                        yy = GT2(r2, t2 - t0z2_l);
                }
        } else {
                r1 = x - t0z1;
                r2 = CHOPPED((r1 - t0z1_l));
                t2 = r1 - r2;
                yy = GT1(r2, t2 - t0z1_l);
        }
        /* compute gamma(x+i) = (x+i-1)*...*(x+1)*yy, 0<i<8 */
        switch (i) {
        case 0:         /* yy/x */
```



```
                r1 = one / x;
                xh = CHOPPED((x));      /* x is not tiny */
                rr.h = CHOPPED(((yy.h + yy.l) * r1));
                rr.l = r1 * (yy.h - rr.h * xh) - ((r1 * rr.h) * (x - xh) -
                        r1 * yy.l);
                break;
        case 1:         /* yy */
                rr.h = yy.h;
                rr.l = yy.l;
                break;
        case 2:         /* (x+1)*yy */
                z = x + one;    /* may not be exact */
                zh = CHOPPED((z));
                rr.h = zh * yy.h;
                rr.l = z * yy.l + (x - (zh - one)) * yy.h;
                break;
        case 3:         /* (x+2)*(x+1)*yy */
                z1 = x + one;
                z2 = x + 2.0L;
                z = z1 * z2;
                xh = CHOPPED((z));
                zh = CHOPPED((z1));
                xl = (x - (zh - one)) * (z2 + zh) - (xh - zh * (zh + one));

                rr.h = xh * yy.h;
                rr.l = z * yy.l + xl * yy.h;
                break;

        case 4:         /* (x+1)*(x+3)*(x+2)*yy */
                z1 = x + 2.0L;
                z2 = (x + one) * (x + 3.0L);
                zh = CHOPPED(z1);
                zl = x - (zh - 2.0L);
                xh = CHOPPED(z2);
                xl = zl * (zh + z1) - (xh - (zh * zh - one));

                /* wh+wl=(x+2)*yy */
                wh = CHOPPED((z1 * (yy.h + yy.l)));
                wl = (zl * yy.h + z1 * yy.l) - (wh - zh * yy.h);

                rr.h = xh * wh;
                rr.l = z2 * wl + xl * wh;

                break;
        case 5:         /* ((x+1)*(x+4)*(x+2)*(x+3))*yy */
                z1 = x + 2.0L;
                z2 = x + 3.0L;
                z = z1 * z2;
                zh = CHOPPED((z1));
                yh = CHOPPED((z));
                yl = (x - (zh - 2.0L)) * (z2 + zh) - (yh - zh * (zh + one));
                z2 = z - 2.0L;
                z *= z2;
                xh = CHOPPED((z));
                xl = yl * (z2 + yh) - (xh - yh * (yh - 2.0L));
                rr.h = xh * yy.h;
                rr.l = z * yy.l + xl * yy.h;
                break;
        case 6:         /* ((x+1)*(x+2)*(x+3)*(x+4)*(x+5))*yy */
                z1 = x + 2.0L;
                z2 = x + 3.0L;
                z = z1 * z2;
                zh = CHOPPED((z1));
                yh = CHOPPED((z));
                z1 = x - (zh - 2.0L);
                yl = z1 * (z2 + zh) - (yh - zh * (zh + one));
```

```
 975                 z2 = z - 2.0L;
 976                 x5 = x + 5.0L;
 977                 z *= z2;
 978                 xh = CHOPPED(z);
 979                 zh += 3.0;
 980                 xl = yl * (z2 + yh) - (xh - yh * (yh - 2.0L));
 981                                         /* xh+xl=(x+1)*...*(x+4) */
 982                 /* wh+wl=(x+5)*yy */
 983                 wh = CHOPPED((x5 * (yy.h + yy.l)));
 984                 wl = (z1 * yy.h + x5 * yy.l) - (wh - zh * yy.h);
 985                 rr.h = wh * xh;
 986                 rr.l = z * wl + xl * wh;
 987                 break;
 988         case 7:         /* ((x+1)*(x+2)*(x+3)*(x+4)*(x+5)*(x+6))*yy */
 989                 z1 = x + 3.0L;
 990                 z2 = x + 4.0L;
 991                 z = z2 * z1;
 992                 zh = CHOPPED((z1));
 993                 yh = CHOPPED((z));      /* yh+yl = (x+3)(x+4) */
 994                 yl = (x - (zh - 3.0L)) * (z2 + zh) - (yh - (zh * (zh + one)));
 995                 z1 = x + 6.0L;
 996                 z2 = z - 2.0L;  /* z2 = (x+2)*(x+5) */
 997                 z *= z2;
 998                 xh = CHOPPED((z));
 999                 xl = yl * (z2 + yh) - (xh - yh * (yh - 2.0L));
1000                                         /* xh+xl=(x+2)*...*(x+5) */
1001                 /* wh+wl=(x+1)(x+6)*yy */
1002                 z2 -= 4.0L;     /* z2 = (x+1)(x+6) */
1003                 wh = CHOPPED((z2 * (yy.h + yy.l)));
1004                 wl = (z2 * yy.l + yl * yy.h) - (wh - (yh - 6.0L) * yy.h);
1005                 rr.h = wh * xh;
1006                 rr.l = z * wl + xl * wh;
1007         }
1008         return (rr);
1009 }

1011 long double
1012 tgammal(long double x) {
1013         struct LDouble ss, ww;
1014         long double t, t1, t2, t3, t4, t5, w, y, z, z1, z2, z3, z5;
1015         int i, j, m, ix, hx, xk;
1016         unsigned lx;

1018         hx = H0_WORD(x);
1019         lx = H3_WORD(x);
1020         ix = hx & 0x7fffffff;
1021         y = x;
1022         if (ix < 0x3f8e0000) {  /* x < 2**-113 */
1023                 return (one / x);
1024         }
1025         if (ix >= 0x7fff0000)
1026                 return (x * ((hx < 0)? zero : x));      /* Inf or NaN */
1027         if (x > overflow)       /* overflow threshold */
1028                 return (x * 1.0e4932L);
1029         if (hx >= 0x40020000) { /* x >= 8 */
1030                 ww = large_gam(x, &m);
1031                 w = ww.h + ww.l;
1032                 return (scalbnl(w, m));
1033         }

1035         if (hx > 0) {           /* 0 < x < 8 */
1035         if (hx > 0) {           /* x from 0 to 8 */
1036                 i = (int) x;
1037                 ww = gam_n(i, x - (long double) i);
1038                 return (ww.h + ww.l);
1039         }
```



```
1040         /* INDENT OFF */
1041         /* negative x */
1042         /*
1043          * compute xk =
1044          *      -2 ... x is an even int (-inf is considered an even #)
1045          *      -1 ... x is an odd int
1046          *      +0 ... x is not an int but chopped to an even int
1047          *      +1 ... x is not an int but chopped to an odd int
1048          */
1049         /* INDENT ON */
1050         xk = 0;
1051 #if defined(__x86)
1052         if (ix >= 0x403e0000) { /* x >= 2**63 } */
1053                 if (ix >= 0x403f0000)
1054                         xk = -2;
1055                 else
1056                         xk = -2 + (lx & 1);
1057 #else
1058         if (ix >= 0x406f0000) { /* x >= 2**112 */
1059                 if (ix >= 0x40700000)
1060                         xk = -2;
1061                 else
1062                         xk = -2 + (lx & 1);
1063 #endif
1064         } else if (ix >= 0x3fff0000) {
1065                 w = -x;
1066                 t1 = floorl(w);
1067                 t2 = t1 * half;
1068                 t3 = floorl(t2);
1069                 if (t1 == w) {
1070                         if (t2 == t3)
1071                                 xk = -2;
1072                         else
1073                                 xk = -1;
1074                 } else {
1075                         if (t2 == t3)
1076                                 xk = 0;
1077                         else
1078                                 xk = 1;
1079                 }
1080         }

1082         if (xk < 0) {
1083                 /* return NaN. Ideally gamma(-n)= (-1)**(n+1) * inf */
1084                 return (x - x) / (x - x);
1085         }

1087         /*
1088          * negative underflow thresold -(1774+9ulp)
1089          */
1090         if (x < -1774.00000000000000000000000000017749370L) {
1091                 z = tiny / x;
1092                 if (xk == 1)
1093                         z = -z;
1094                 return (z * tiny);
1095         }

1097         /* INDENT OFF */
1098         /*
1099          * now compute gamma(x) by  -1/((sin(pi*y)/pi)*gamma(1+y)), y = -x
1100          */
1101         /*
1102          * First compute ss = -sin(pi*y)/pi so that
1103          * gamma(x) = 1/(ss*gamma(1+y))
1104          */
1105         /* INDENT ON */
```

```
1106         y = -x;
1107         j = (int) y;
1108         z = y - (long double) j;
1109         if (z > 0.318309886183790671537767 5L)
1110                 if (z > 0.681690113816209328462232 5L)
1111                         ss = kpsin(one - z);
1112                 else
1113                         ss = kpcos(0.5L - z);
1114         else
1115                 ss = kpsin(z);
1116         if (xk == 0) {
1117                 ss.h = -ss.h;
1118                 ss.l = -ss.l;
1119         }

1121         /* Then compute ww = gamma(1+y), note that result scale to 2**m */
1122         m = 0;
1123         if (j < 7) {
1124                 ww = gam_n(j + 1, z);
1125         } else {
1126                 w = y + one;
1127                 if ((lx & 1) == 0) {    /* y+1 exact (note that y<184) */
1128                         ww = large_gam(w, &m);
1129                 } else {
1130                         t = w - one;
1131                         if (t == y) {   /* y+one exact */
1132                                 ww = large_gam(w, &m);
1133                         } else {        /* use y*gamma(y) */
1134                                 if (j == 7)
1135                                         ww = gam_n(j, z);
1136                                 else
1137                                         ww = large_gam(y, &m);
1138                                 t4 = ww.h + ww.l;
1139                                 t1 = CHOPPED((y));
1140                                 t2 = CHOPPED((t4));
1141                                                 /* t4 will not be too large */
1142                                 ww.l = y * (ww.l - (t2 - ww.h)) + (y - t1) * t2;
1143                                 ww.h = t1 * t2;
1144                         }
1145                 }
1146         }

1148         /* compute 1/(ss*ww) */
1149         t3 = ss.h + ss.l;
1150         t4 = ww.h + ww.l;
1151         t1 = CHOPPED((t3));
1152         t2 = CHOPPED((t4));
1153         z1 = ss.l - (t1 - ss.h);        /* (t1,z1) = ss */
1154         z2 = ww.l - (t2 - ww.h);        /* (t2,z2) = ww */
1155         t3 = t3 * t4;                   /* t3 = ss*ww */
1156         z3 = one / t3;                  /* z3 = 1/(ss*ww) */
1157         t5 = t1 * t2;
1158         z5 = z1 * t4 + t1 * z2;         /* (t5,z5) = ss*ww */
1159         t1 = CHOPPED((t3));             /* (t1,z1) = ss*ww */
1160         z1 = z5 - (t1 - t5);
1161         t2 = CHOPPED((z3));             /* leading 1/(ss*ww) */
1162         z2 = z3 * (t2 * z1 - (one - t2 * t1));
1163         z = t2 - z2;

1165         return (scalbnl(z, -m));
1166 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
    5808 Sun May 11 12:16:25 2014
new/usr/src/lib/libmvec/Makefile.com
**********************************************************
#
# 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 \
                  __vatanf.o \
                  __vcos.o \
                  __vcosf.o \
                  __vexp.o \
                  __vexpf.o \
                  __vhypot.o \
                  __vhypotf.o \
                  __vlog.o \
                  __vlogf.o \
                  __vpow.o \
                  __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 CERRWARN        += -_gcc=-Wno-parentheses
 174 CERRWARN        += -_gcc=-Wno-uninitialized
 174 CERRWARN        += -_gcc=-Wno-unused-variable

 176 LINTFLAGS       += $(LINTERROFF)
 177 LINTFLAGS64     += $(LINTERROFF)
 178 LINTFLAGS64     += -errchk=longptr64

 180 CLAGS           += $(LINTERROFF)
 181 CFLAGS64        += $(LINTERROFF)

 183 ASDEF           += -DLIBMVEC_SO_BUILD

 185 FLTRPATH_sparc          = $$ORIGIN/cpu/$$ISALIST/libmvec_isa.so.1
 186 FLTRPATH_sparcv9        = $$ORIGIN/../cpu/$$ISALIST/sparcv9/libmvec_isa.so.1
 187 FLTRPATH_i386           = $$ORIGIN/libmvec/$$HWCAP
 188 FLTRPATH                = $(FLTRPATH_$(TARGET_ARCH))

 190 sparc_CFLAGS += -_cc=-W0,-xintrinsic
 191 sparcv9_CFLAGS += -_cc=-W0,-xintrinsic
 192 CPPFLAGS_i386   += -Dfabs=__fabs
```



```
 194 CPPFLAGS        += -DLIBMVEC_SO_BUILD

 196 SRCS_mvec_i386 = \
 197         ../common/__vsqrtf.c \
 198         #end

 200 SRCS_mvec_sparc = \
 201         $(SRCS_mvec_i386) \
 202         #end
 203 SRCS_mvec_sparcv9 = \
 204         $(SRCS_mvec_i386) \
 205         #end

 207 SRCS_mvec = \
 208         $(SRCS_mvec_$(TARGETMACH)) \
 209         ../common/__vTBL_atan1.c \
 210         ../common/__vTBL_atan2.c \
 211         ../common/__vTBL_rsqrt.c \
 212         ../common/__vTBL_sincos.c \
 213         ../common/__vTBL_sincos2.c \
 214         ../common/__vTBL_sqrtf.c \
 215         ../common/__vatan.c \
 216         ../common/__vatan2.c \
 217         ../common/__vatan2f.c \
 218         ../common/__vatanf.c \
 219         ../common/__vc_abs.c \
 220         ../common/__vc_exp.c \
 221         ../common/__vc_log.c \
 222         ../common/__vc_pow.c \
 223         ../common/__vcos.c \
 224         ../common/__vcosbig.c \
 225         ../common/__vcosbigf.c \
 226         ../common/__vcosf.c \
 227         ../common/__vexp.c \
 228         ../common/__vexpf.c \
 229         ../common/__vhypot.c \
 230         ../common/__vhypotf.c \
 231         ../common/__vlog.c \
 232         ../common/__vlogf.c \
 233         ../common/__vpow.c \
 234         ../common/__vpowf.c \
 235         ../common/__vrem_pio2m.c \
 236         ../common/__vrhypot.c \
 237         ../common/__vrhypotf.c \
 238         ../common/__vrsqrt.c \
 239         ../common/__vrsqrtf.c \
 240         ../common/__vsin.c \
 241         ../common/__vsinbig.c \
 242         ../common/__vsinbigf.c \
 243         ../common/__vsincos.c \
 244         ../common/__vsincosbig.c \
 245         ../common/__vsincosbigf.c \
 246         ../common/__vsincosf.c \
 247         ../common/__vsinf.c \
 248         ../common/__vsqrt.c \
 249         ../common/__vz_abs.c \
 250         ../common/__vz_exp.c \
 251         ../common/__vz_log.c \
 252         ../common/__vz_pow.c \
 253         ../common/vatan2_.c \
 254         ../common/vatan2f_.c \
 255         ../common/vatan_.c \
 256         ../common/vatanf_.c \
 257         ../common/vc_abs_.c \
 258         ../common/vc_exp_.c \
 259         ../common/vc_log_.c \
```

```
260         ../common/vc_pow_.c \
261         ../common/vcos_.c \
262         ../common/vcosf_.c \
263         ../common/vexp_.c \
264         ../common/vexpf_.c \
265         ../common/vhypot_.c \
266         ../common/vhypotf_.c \
267         ../common/vlog_.c \
268         ../common/vlogf_.c \
269         ../common/vpow_.c \
270         ../common/vpowf_.c \
271         ../common/vrhypot_.c \
272         ../common/vrhypotf_.c \
273         ../common/vrsqrt_.c \
274         ../common/vrsqrtf_.c \
275         ../common/vsin_.c \
276         ../common/vsincos_.c \
277         ../common/vsincosf_.c \
278         ../common/vsinf_.c \
279         ../common/vsqrt_.c \
280         ../common/vsqrtf_.c \
281         ../common/vz_abs_.c \
282         ../common/vz_exp_.c \
283         ../common/vz_log_.c \
284         ../common/vz_pow_.c \
285         #end

287 .KEEP_STATE:

289 all:    $(LIBS)

291 lint:   lintcheck

293 pics/%.o: ../$(TARGET_ARCH)/src/%.S
294         $(COMPILE.s) -o $@ $<
295         $(POST_PROCESS_O)

297 pics/%.o: ../common/$$(CHIP)/%.S
298         $(COMPILE.s) -o $@ $<
299         $(POST_PROCESS_O)
```

```
**********************************************************
   11255 Sun May 11 12:16:27 2014
new/usr/src/lib/libmvec/common/__vatan.c
**********************************************************
```

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

void
__vatan( int n, double * restrict x, int stridex, double * restrict y, int strid
{
  double  f , z, ans = 0.0L, ansu , ansl , tmp , poly , conup , conlo , dummy;
  double  f , z, ans, ansu , ansl , tmp , poly , conup , conlo , dummy;
  double  f1,   ans1, ansu1, ansl1, tmp1, poly1, conup1, conlo1;
  double  f2,   ans2, ansu2, ansl2, tmp2, poly2, conup2, conlo2;
  int index, sign, intf, intflo, intz, argcount;
  int index1, sign1 = 0;
  int index2, sign2 = 0;
  double *yaddr,*yaddr1 = 0,*yaddr2 = 0;
  int index1, sign1 ;
  int index2, sign2 ;
  double *yaddr,*yaddr1,*yaddr2;
  extern const double __vlibm_TBL_atan1[];
  extern double fabs( double );
```



```
/*    Power series  atan(x) = x + p1*x**3 + p2*x**5 + p3*x**7
 *    Error =  -3.08254E-18   On the interval  |x| < 1/64 */

/* define dummy names for readability.  Use parray to help compiler optimize loa
#define p3    parray[0]
#define p2    parray[1]
#define p1    parray[2]

  static const double parray[] = {
   -1.42802904684429972E-01,          /* p[3]         */
    1.99999991724700061E-01,          /* p[2]         */
   -3.33333333329292858E-01,          /* p[1]         */
    1.0,                              /* not used for p[0], though
   -1.0,                              /* used to flip sign of answer
  };

  if( n <= 0 ) return;          /* if no. of elements is 0 or neg, do nothing */
  do
  {
  LOOP0:

    f        = fabs(*x);                        /* fetch argument
    intf     = HI(x);                   /* upper half of x, as integer  */
    intflo   = LO(x);                   /* lower half of x, as integer  */
    sign     = intf &  0x80000000;              /* sign of argument
    intf     = intf & ~0x80000000;              /* abs(upper argument)

    if( (intf > 0x43600000) || (intf < 0x3e300000) ) /* filter out special cases
    {
      if(  (intf > 0x7ff00000) || ((intf == 0x7ff00000) &&  (intflo !=0) ) )
      {
        ans   = f - f;                          /* return NaN if x=NaN*/
      }
      else if( intf < 0x3e300000 )              /* avoid underflow for small arg
      {
        dummy = 1.0e37 + f;
        dummy = dummy;
        ans   = f;
      }
      else if( intf > 0x43600000 )              /* avoid underflow for big arg
      {
        index = 2;
        ans   = __vlibm_TBL_atan1[index] + __vlibm_TBL_atan1[index+1];/* pi/2 up
      }
      *y      = (sign) ? -ans: ans;             /* store answer, with sign bit
      x      += stridex;
      y      += stridey;
      argcount = 0;                             /* initialize argcount
      if ( --n <=0 ) break;                     /* we are done
      goto LOOP0;                               /* otherwise, examine next arg
    }

    index    = 0;                               /* points to 0,0 in table
    if (intf > 0x40500000)                      /* if(|x| > 64
    { f = -1.0/f;
      index  = 2;                               /* point to pi/2 upper, lower
    }
    else if( intf >= 0x3f900000 )               /* if |x| >= (1/64)...
    {
      intz   = (intf + 0x00008000) & 0x7fff0000;/* round arg, keep upper
      HI(&z)  = intz;                           /* store as a double (z)
      LO(&z)  = 0;                              /* ...lower
      f      = (f - z)/(1.0 + f*z);             /* get reduced argument
      index  = (intz - 0x3f900000) >> 15;       /* (index >> 16) << 1)
      index  = index + 4;                       /* skip over 0,0,pi/2,pi/2
```

```
125     }
126     yaddr    = y;                                /* address to store this answer
127     x       += stridex;                          /* point to next arg
128     y       += stridey;                          /* point to next result
129     argcount = 1;                                /* we now have 1 good argument
130     if ( --n <=0 )
131     {
132       f1      = 0.0;                             /* put dummy values in args 1,2
133       f2      = 0.0;
134       index1  = 0;
135       index2  = 0;
136       goto UNROLL3;                              /* finish up with 1 good arg
137     }

139     /*--------------------------------------------------------------------------
140     /*--------------------------------------------------------------------------
141     /*--------------------------------------------------------------------------

143   LOOP1:

145     f1       = fabs(*x);                         /* fetch argument
146     intf     = HI(x);                  /* upper half of x, as integer  */
147     intflo   = LO(x);                  /* lower half of x, as integer  */
148     sign1    = intf &  0x80000000;               /* sign of argument
149     intf     = intf & ~0x80000000;               /* abs(upper argument)
150
151     if( (intf > 0x43600000) || (intf < 0x3e300000) ) /* filter out special cases
152     {
153       if(  (intf > 0x7ff00000) || ((intf == 0x7ff00000) &&  (intflo !=0) ) )
154       {
155         ans   = f1 - f1;                         /* return NaN if x=NaN*/
156       }
157       else if( intf < 0x3e300000 )               /* avoid underflow for small arg
158       {
159         dummy = 1.0e37 + f1;
160         dummy = dummy;
161         ans   = f1;
162       }
163       else if( intf > 0x43600000 )               /* avoid underflow for big arg
164       {
165         index1 = 2;
166         ans   = __vlibm_TBL_atan1[index1] + __vlibm_TBL_atan1[index1+1];/* pi/2
167       }
168       *y      = (sign1) ? -ans: ans;             /* store answer, with sign bit
169       x      += stridex;
170       y      += stridey;
171       argcount = 1;                              /* we still have 1 good arg
172       if ( --n <=0 )
173       {
174         f1      = 0.0;                           /* put dummy values in args 1,2
175         f2      = 0.0;
176         index1  = 0;
177         index2  = 0;
178         goto UNROLL3;                            /* finish up with 1 good arg
179       }
180       goto LOOP1;                                /* otherwise, examine next arg
181     }
182
183     index1   = 0;                                /* points to 0,0 in table
184     if (intf > 0x40500000)                       /* if(|x| > 64
185     { f1 = -1.0/f1;
186       index1 = 2;                                /* point to pi/2 upper, lower
187     }
188     else if( intf >= 0x3f900000 )                /* if |x| >= (1/64)...
189     {
190       intz   = (intf + 0x00008000) & 0x7fff0000;/* round arg, keep upper
```



```
191       HI(&z) = intz;                             /* store as a double (z)
192       LO(&z) = 0;                                /* ...lower
193       f1     = (f1 - z)/(1.0 + f1*z);            /* get reduced argument
194       index1 = (intz - 0x3f900000) >> 15;        /* (index >> 16) << 1)
195       index1 = index1 + 4;                       /* skip over 0,0,pi/2,pi/2
196     }
197     yaddr1   = y;                                /* address to store this answer
198     x       += stridex;                          /* point to next arg
199     y       += stridey;                          /* point to next result
200     argcount = 2;                                /* we now have 2 good arguments
201     if ( --n <=0 )
202     {
203       f2      = 0.0;                             /* put dummy value in arg 2 */
204       index2  = 0;
205       goto UNROLL3;                              /* finish up with 2 good args
206     }

208     /*--------------------------------------------------------------------------
209     /*--------------------------------------------------------------------------
210     /*--------------------------------------------------------------------------

212   LOOP2:

214     f2       = fabs(*x);                         /* fetch argument
215     intf     = HI(x);                  /* upper half of x, as integer  */
216     intflo   = LO(x);                  /* lower half of x, as integer  */
217     sign2    = intf &  0x80000000;               /* sign of argument
218     intf     = intf & ~0x80000000;               /* abs(upper argument)
219
220     if( (intf > 0x43600000) || (intf < 0x3e300000) ) /* filter out special cases
221     {
222       if(  (intf > 0x7ff00000) || ((intf == 0x7ff00000) &&  (intflo !=0) ) )
223       {
224         ans   = f2 - f2;                         /* return NaN if x=NaN*/
225       }
226       else if( intf < 0x3e300000 )               /* avoid underflow for small arg
227       {
228         dummy = 1.0e37 + f2;
229         dummy = dummy;
230         ans   = f2;
231       }
232       else if( intf > 0x43600000 )               /* avoid underflow for big arg
233       {
234         index2 = 2;
235         ans   = __vlibm_TBL_atan1[index2] + __vlibm_TBL_atan1[index2+1];/* pi/2
236       }
237       *y      = (sign2) ? -ans: ans;             /* store answer, with sign bit
238       x      += stridex;
239       y      += stridey;
240       argcount = 2;                              /* we still have 2 good args
241       if ( --n <=0 )
242       {
243         f2      = 0.0;                           /* put dummy value in arg 2 */
244         index2  = 0;
245         goto UNROLL3;                            /* finish up with 2 good args
246       }
247       goto LOOP2;                                /* otherwise, examine next arg
248     }
249
250     index2   = 0;                                /* points to 0,0 in table
251     if (intf > 0x40500000)                       /* if(|x| > 64
252     { f2 = -1.0/f2;
253       index2 = 2;                                /* point to pi/2 upper, lower
254     }
255     else if( intf >= 0x3f900000 )                /* if |x| >= (1/64)...
256     {
```

```
 257       intz   = (intf + 0x00008000) & 0x7fff0000;/* round arg, keep upper
 258       HI(&z) = intz;                           /* store as a double (z)
 259       LO(&z) = 0;                              /* ...lower
 260       f2     = (f2 - z)/(1.0 + f2*z);          /* get reduced argument
 261       index2 = (intz - 0x3f900000) >> 15;      /* (index >> 16) << 1)
 262       index2 = index2 + 4;                     /* skip over 0,0,pi/2,pi/2
 263     }
 264     yaddr2   = y;                              /* address to store this answer
 265     x       += stridex;                        /* point to next arg
 266     y       += stridey;                        /* point to next result
 267     argcount = 3;                              /* we now have 3 good arguments


 270 /* here is the 3 way unrolled section,
 271    note, we may actually only have
 272    1,2, or 3 'real' arguments at this point
 273 */

 275 UNROLL3:

 277     conup    = __vlibm_TBL_atan1[index ];      /* upper table
 278     conup1   = __vlibm_TBL_atan1[index1];      /* upper table
 279     conup2   = __vlibm_TBL_atan1[index2];      /* upper table

 281     conlo    = __vlibm_TBL_atan1[index +1];    /* lower table
 282     conlo1   = __vlibm_TBL_atan1[index1+1];    /* lower table
 283     conlo2   = __vlibm_TBL_atan1[index2+1];    /* lower table

 285     tmp      = f *f ;
 286     tmp1     = f1*f1;
 287     tmp2     = f2*f2;

 289     poly     = f *((p3*tmp  + p2)*tmp  + p1)*tmp ;
 290     poly1    = f1*((p3*tmp1 + p2)*tmp1 + p1)*tmp1;
 291     poly2    = f2*((p3*tmp2 + p2)*tmp2 + p1)*tmp2;

 293     ansu     = conup  + f ;                    /* compute atan(f)  upper
 294     ansu1    = conup1 + f1;                    /* compute atan(f)  upper
 295     ansu2    = conup2 + f2;                    /* compute atan(f)  upper

 297     ansl     = (((conup  - ansu ) + f ) + poly ) + conlo ;
 298     ansl1    = (((conup1 - ansu1) + f1) + poly1) + conlo1;
 299     ansl2    = (((conup2 - ansu2) + f2) + poly2) + conlo2;

 301     ans      = ansu  + ansl ;
 302     ans1     = ansu1 + ansl1;
 303     ans2     = ansu2 + ansl2;

 305 /* now check to see if these are 'real' or 'dummy' arguments BEFORE storing */

 307    *yaddr    = sign ? -ans: ans;               /* this one is always good
 308    if(argcount < 3) break;                    /* end loop and finish up
 309      *yaddr1   = sign1 ? -ans1: ans1;
 310      *yaddr2   = sign2 ? -ans2: ans2;

 312   }  while (--n > 0);

 314  if(argcount == 2)
 315    {  *yaddr1  = sign1 ? -ans1: ans1;
 316    }
 317 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
    8571 Sun May 11 12:16:28 2014
new/usr/src/lib/libmvec/common/__vatan2f.c
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" 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.
 */

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

extern const double __vlibm_TBL_atan1[];

static const double
pio4    =  7.8539816339744827900e-01,
pio2    =  1.5707963267948965580e+00,
pi      =  3.1415926535897931160e+00;

static const float
zero    =  0.0f,
one     =  1.0f,
q1      = -3.3333333333296428046e-01f,
q2      =  1.9999999186853752618e-01f,
twop24  =  16777216.0f;

void
__vatan2f( int n, float * restrict y, int stridey, float * restrict x,
        int stridex, float * restrict z, int stridez )
{
        float           x0, x1, x2, y0, y1, y2, *pz0 = 0, *pz1, *pz2;
        float           x0, x1, x2, y0, y1, y2, *pz0, *pz1, *pz2;
        double          ah0, ah1, ah2;
        double          t0, t1, t2;
        double          sx0, sx1, sx2;
        double          sign0, sign1, sign2;
        int             i, k0 = 0, k1, k2, hx, sx, sy;
        int             i, k0, k1, k2, hx, sx, sy;
        int             hy0, hy1, hy2;
```



```
        float           base0 = 0.0, base1, base2;
        float           base0, base1, base2;
        double          num0, num1, num2;
        double          den0, den1, den2;
        double          dx0, dx1, dx2;
        double          dy0, dy1, dy2;
        double          db0, db1, db2;

        do
        {
loop0:
                hy0 = *(int*)y;
                hx = *(int*)x;
                sign0 = one;
                sy = hy0 & 0x80000000;
                hy0 &= ~0x80000000;

                sx = hx & 0x80000000;
                hx &= ~0x80000000;

                if ( hy0 > hx )
                {
                        x0 = *y;
                        y0 = *x;
                        i = hx;
                        hx = hy0;
                        hy0 = i;
                        if ( sy )
                        {
                                x0 = -x0;
                                sign0 = -sign0;
                        }
                        if ( sx )
                        {
                                y0 = -y0;
                                ah0 = pio2;
                        }
                        else
                        {
                                ah0 = -pio2;
                                sign0 = -sign0;
                        }
                }
                else
                {
                        y0 = *y;
                        x0 = *x;
                        if ( sy )
                        {
                                y0 = -y0;
                                sign0 = -sign0;
                        }
                        if ( sx )
                        {
                                x0 = -x0;
                                ah0 = -pi;
                                sign0 = -sign0;
                        }
                        else
                                ah0 = zero;
                }

                if ( hx >= 0x7f800000 || hx - hy0 >= 0x0c800000 )
                {
                        if ( hx >= 0x7f800000 )
                        {
```

```
126                                 if ( hx ^ 0x7f800000 ) /* nan */
127                                         ah0 =  x0 + y0;
128                                 else if ( hy0 >= 0x7f800000 )
129                                         ah0 += pio4;
130                         }
131                         else if ( (int) ah0 == 0 )
132                                 ah0 = y0 / x0;
133                         *z = (sign0 == one) ? ah0 : -ah0;
134 /* sign0*ah0 would change nan behavior relative to previous release */
135                         x += stridex;
136                         y += stridey;
137                         z += stridez;
138                         i = 0;
139                         if ( --n <= 0 )
140                                 break;
141                         goto loop0;
142                 }
143                 if (hy0 < 0x00800000) {
144                         if ( hy0 == 0 )
145                         {
146                                 *z = sign0 * (float) ah0;
147                                 x += stridex;
148                                 y += stridey;
149                                 z += stridez;
150                                 i = 0;
151                                 if ( --n <= 0 )
152                                         break;
153                                 goto loop0;
154                         }
155                         y0 *= twop24; /* scale subnormal y */
156                         x0 *= twop24; /* scale possibly subnormal x */
157                         hy0 = *(int*)&y0;
158                         hx = *(int*)&x0;
159                 }
160                 pz0 = z;

162                 k0 = ( hy0 - hx + 0x3f800000 ) & 0xfff80000;
163                 if( k0 >= 0x3C800000 )          /* if |x| >= (1/64)... */
164                 {
165                         *(int*)&base0 = k0;
166                         k0 = (k0 - 0x3C800000) >> 18; /* (index >> 19) << 1) */
167                         k0 += 4;
168                                 /* skip over 0,0,pi/2,pi/2 */
169                 }
170                 else                            /* |x| < 1/64 */
171                 {
172                         k0 = 0;
173                         base0 = zero;
174                 }

176                 x += stridex;
177                 y += stridey;
178                 z += stridez;
179                 i = 1;
180                 if ( --n <= 0 )
181                         break;


184 loop1:
185                 hy1 = *(int*)y;
186                 hx = *(int*)x;
187                 sign1 = one;
188                 sy = hy1 & 0x80000000;
189                 hy1 &= ~0x80000000;

191                 sx = hx & 0x80000000;
```



```
192                 hx &= ~0x80000000;

194                 if ( hy1 > hx )
195                 {
196                         x1 = *y;
197                         y1 = *x;
198                         i = hx;
199                         hx = hy1;
200                         hy1 = i;
201                         if ( sy )
202                         {
203                                 x1 = -x1;
204                                 sign1 = -sign1;
205                         }
206                         if ( sx )
207                         {
208                                 y1 = -y1;
209                                 ah1 = pio2;
210                         }
211                         else
212                         {
213                                 ah1 = -pio2;
214                                 sign1 = -sign1;
215                         }
216                 }
217                 else
218                 {
219                         y1 = *y;
220                         x1 = *x;
221                         if ( sy )
222                         {
223                                 y1 = -y1;
224                                 sign1 = -sign1;
225                         }
226                         if ( sx )
227                         {
228                                 x1 = -x1;
229                                 ah1 = -pi;
230                                 sign1 = -sign1;
231                         }
232                         else
233                                 ah1 = zero;
234                 }

236                 if ( hx >= 0x7f800000 || hx - hy1 >= 0x0c800000 )
237                 {
238                         if ( hx >= 0x7f800000 )
239                         {
240                                 if ( hx ^ 0x7f800000 ) /* nan */
241                                         ah1 =  x1 + y1;
242                                 else if ( hy1 >= 0x7f800000 )
243                                         ah1 += pio4;
244                         }
245                         else if ( (int) ah1 == 0 )
246                                 ah1 = y1 / x1;
247                         *z = (sign1 == one)? ah1 : -ah1;
248                         x += stridex;
249                         y += stridey;
250                         z += stridez;
251                         i = 1;
252                         if ( --n <= 0 )
253                                 break;
254                         goto loop1;
255                 }
256                 if (hy1 < 0x00800000) {
257                         if ( hy1 == 0 )
```

```
258                         {
259                                 *z = sign1 * (float) ah1;
260                                 x += stridex;
261                                 y += stridey;
262                                 z += stridez;
263                                 i = 1;
264                                 if ( --n <= 0 )
265                                         break;
266                                 goto loop1;
267                         }
268                         y1 *= twop24; /* scale subnormal y */
269                         x1 *= twop24; /* scale possibly subnormal x */
270                         hy1 = *(int*)&y1;
271                         hx = *(int*)&x1;
272                 }
273                 pz1 = z;

275                 k1 = ( hy1 - hx + 0x3f800000 ) & 0xfff80000;
276                 if( k1 >= 0x3C800000 )          /* if |x| >= (1/64)... */
277                 {
278                         *(int*)&base1 = k1;
279                         k1 = (k1 - 0x3C800000) >> 18; /* (index >> 19) << 1) */
280                         k1 += 4;
281                                 /* skip over 0,0,pi/2,pi/2 */
282                 }
283                 else                            /* |x| < 1/64 */
284                 {
285                         k1 = 0;
286                         base1 = zero;
287                 }

289                 x += stridex;
290                 y += stridey;
291                 z += stridez;
292                 i = 2;
293                 if ( --n <= 0 )
294                         break;

296 loop2:
297                 hy2 = *(int*)y;
298                 hx = *(int*)x;
299                 sign2 = one;
300                 sy = hy2 & 0x80000000;
301                 hy2 &= ~0x80000000;

303                 sx = hx & 0x80000000;
304                 hx &= ~0x80000000;

306                 if ( hy2 > hx )
307                 {
308                         x2 = *y;
309                         y2 = *x;
310                         i = hx;
311                         hx = hy2;
312                         hy2 = i;
313                         if ( sy )
314                         {
315                                 x2 = -x2;
316                                 sign2 = -sign2;
317                         }
318                         if ( sx )
319                         {
320                                 y2 = -y2;
321                                 ah2 = pio2;
322                         }
323                         else
```



```
324                         {
325                                 ah2 = -pio2;
326                                 sign2 = -sign2;
327                         }
328                 }
329                 else
330                 {
331                         y2 = *y;
332                         x2 = *x;
333                         if ( sy )
334                         {
335                                 y2 = -y2;
336                                 sign2 = -sign2;
337                         }
338                         if ( sx )
339                         {
340                                 x2 = -x2;
341                                 ah2 = -pi;
342                                 sign2 = -sign2;
343                         }
344                         else
345                                 ah2 = zero;
346                 }

348                 if ( hx >= 0x7f800000 || hx - hy2 >= 0x0c800000 )
349                 {
350                         if ( hx >= 0x7f800000 )
351                         {
352                                 if ( hx ^ 0x7f800000 ) /* nan */
353                                         ah2 =  x2 + y2;
354                                 else if ( hy2 >= 0x7f800000 )
355                                         ah2 += pio4;
356                         }
357                         else if ( (int) ah2 == 0 )
358                                 ah2 = y2 / x2;
359                         *z = (sign2 == one)? ah2 : -ah2;
360                         x += stridex;
361                         y += stridey;
362                         z += stridez;
363                         i = 2;
364                         if ( --n <= 0 )
365                                 break;
366                         goto loop2;
367                 }
368                 if (hy2 < 0x00800000) {
369                         if ( hy2 == 0 )
370                         {
371                                 *z = sign2 * (float) ah2;
372                                 x += stridex;
373                                 y += stridey;
374                                 z += stridez;
375                                 i = 2;
376                                 if ( --n <= 0 )
377                                         break;
378                                 goto loop2;
379                         }
380                         y2 *= twop24; /* scale subnormal y */
381                         x2 *= twop24; /* scale possibly subnormal x */
382                         hy2 = *(int*)&y2;
383                         hx = *(int*)&x2;
384                 }

386                 pz2 = z;

388                 k2 = ( hy2 - hx + 0x3f800000 ) & 0xfff80000;
389                 if( k2 >= 0x3C800000 )          /* if |x| >= (1/64)... */
```

```
390                 {
391                         *(int*)&base2 = k2;
392                         k2 = (k2 - 0x3C800000) >> 18; /* (index >> 19) << 1) */
393                         k2 += 4;
394                                 /* skip over 0,0,pi/2,pi/2 */
395                 }
396                 else                            /* |x| < 1/64 */
397                 {
398                         k2 = 0;
399                         base2 = zero;
400                 }

402                 goto endloop;

404 endloop:

406                 ah2 += __vlibm_TBL_atan1[k2];
407                 ah1 += __vlibm_TBL_atan1[k1];
408                 ah0 += __vlibm_TBL_atan1[k0];

410                 db2 = base2;
411                 db1 = base1;
412                 db0 = base0;
413                 dy2 = y2;
414                 dy1 = y1;
415                 dy0 = y0;
416                 dx2 = x2;
417                 dx1 = x1;
418                 dx0 = x0;

420                 num2 = dy2 - dx2 * db2;
421                 den2 = dx2 + dy2 * db2;

423                 num1 = dy1 - dx1 * db1;
424                 den1 = dx1 + dy1 * db1;

426                 num0 = dy0 - dx0 * db0;
427                 den0 = dx0 + dy0 * db0;

429                 t2 = num2 / den2;
430                 t1 = num1 / den1;
431                 t0 = num0 / den0;

433                 sx2 = t2 * t2;
434                 sx1 = t1 * t1;
435                 sx0 = t0 * t0;
436
437                 t2 += t2 * sx2 * ( q1 + sx2 * q2 );
438                 t1 += t1 * sx1 * ( q1 + sx1 * q2 );
439                 t0 += t0 * sx0 * ( q1 + sx0 * q2 );

441                 t2 += ah2;
442                 t1 += ah1;
443                 t0 += ah0;

445                 *pz2 = sign2 * t2;
446                 *pz1 = sign1 * t1;
447                 *pz0 = sign0 * t0;

449                 x += stridex;
450                 y += stridey;
451                 z += stridez;
452                 i = 0;
453         } while ( --n > 0 );

455         if ( i > 1 )
```



```
456         {
457                 ah1 += __vlibm_TBL_atan1[k1];
458                 t1 = ( y1 - x1 * (double)base1 ) /
459                         ( x1 + y1 * (double)base1 );
460                 sx1 = t1 * t1;
461                 t1 += t1 * sx1 * ( q1 + sx1 * q2 );
462                 t1 += ah1;
463                 *pz1 = sign1 * t1;
464         }

466         if ( i > 0 )
467         {
468                 ah0 += __vlibm_TBL_atan1[k0];
469                 t0 = ( y0 - x0 * (double)base0 ) /
470                         ( x0 + y0 * (double)base0 );
471                 sx0 = t0 * t0;
472                 t0 += t0 * sx0 * ( q1 + sx0 * q2 );
473                 t0 += ah0;
474                 *pz0 = sign0 * t0;
475         }
476 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   12272 Sun May 11 12:16:30 2014
new/usr/src/lib/libmvec/common/__vatanf.c
**********************************************************
```

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

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

void
__vatanf( int n, float * restrict x, int stridex, float * restrict y, int stride
{
  extern const double __vlibm_TBL_atan1[];
  double  conup0, conup1, conup2;
  float dummy, ansf = 0.0;
  float dummy, ansf;
  float f0, f1, f2;
  float ans0, ans1, ans2;
  float poly0, poly1, poly2;
  float sign0, sign1, sign2;
  int intf, intz, argcount;
  int index0, index1, index2;
  float z,*yaddr0,*yaddr1,*yaddr2;
  int *pz = (int *) &z;
#ifdef UNROLL4
  double conup3;
  int index3;
  float f3, ans3, poly3, sign3, *yaddr3;
#endif

/*    Power series  atan(x) = x + p1*x**3 + p2*x**5 + p3*x**7
 *    Error =  -3.08254E-18   On the interval  |x| < 1/64 */

  static const float p1 = -0.33329644f /* -3.333333333292928 58E-01f */ ;
  static const float pone = 1.0f;
```



```
  if( n <= 0 ) return;          /* if no. of elements is 0 or neg, do nothing */
  do
  {
  LOOP0:

       intf     = *(int *) x;          /* upper half of x, as integer */
       f0 = *x;
       sign0 = pone;
       if (intf < 0) {
               intf = intf & ~0x80000000; /* abs(upper argument) */
               f0 = -f0;
               sign0 = -sign0;
       }

    if( (intf > 0x5B000000) || (intf < 0x31800000) ) /* filter out special cases
    {
      if( intf > 0x7f800000 )
      {
        ansf  = f0- f0;                                 /* return NaN if x=NaN*/
      }
      else if( intf < 0x31800000 )              /* avoid underflow for small arg
      {
        dummy = 1.0e37 + f0;
        dummy = dummy;
        ansf  = f0;
      }
      else if( intf > 0x5B000000 )              /* avoid underflow for big arg
      {
        index0= 2;
        ansf  = __vlibm_TBL_atan1[index0];/* pi/2 up */
      }
      *y      = sign0*ansf;             /* store answer, with sign bit  */
      x      += stridex;
      y      += stridey;
      argcount = 0;                             /* initialize argcount
      if ( --n <=0 ) break;                     /* we are done
      goto LOOP0;                               /* otherwise, examine next arg
    }

    if (intf > 0x42800000)                      /* if(|x| > 64
    {
    f0 = -pone/f0;
        index0 = 2;                             /* point to pi/2 upper, lower
    }
    else if( intf >= 0x3C800000 )               /* if |x| >= (1/64)...
    {
      intz   = (intf + 0x00040000) & 0x7ff80000;/* round arg, keep upper
      pz[0]  = intz;                            /* store as a float (z)
    f0 = (f0 - z)/(pone + f0*z);
        index0 = (intz - 0x3C800000) >> 18;     /* (index >> 19) << 1)
        index0 = index0+ 4;                     /* skip over 0,0,pi/2,pi/2
    }
    else                                        /* |x| < 1/64 */
    {
        index0   = 0;                           /* points to 0,0 in table
    }
    yaddr0   = y;                               /* address to store this answer
    x       += stridex;                         /* point to next arg
    y       += stridey;                         /* point to next result
    argcount = 1;                               /* we now have 1 good argument
    if ( --n <=0 )
    {
      goto UNROLL;                              /* finish up with 1 good arg
    }

    /*-------------------------------------------------------------------------
```

```
128     /*--------------------------------------------------------------------------
129     /*--------------------------------------------------------------------------

131   LOOP1:

133         intf     = *(int *) x;          /* upper half of x, as integer */
134         f1 = *x;
135         sign1 = pone;
136         if (intf < 0) {
137                 intf = intf & ~0x80000000; /* abs(upper argument) */
138                 f1 = -f1;
139                 sign1 = -sign1;
140         }
141
142     if( (intf > 0x5B000000) || (intf < 0x31800000) ) /* filter out special cases
143     {
144       if( intf > 0x7f800000 )
145       {
146         ansf   = f1 - f1;                       /* return NaN if x=NaN*/
147       }
148       else if( intf < 0x31800000 )              /* avoid underflow for small arg
149       {
150         dummy = 1.0e37 + f1;
151         dummy = dummy;
152         ansf   = f1;
153       }
154       else if( intf > 0x5B000000 )              /* avoid underflow for big arg
155       {
156         index1 = 2;
157         ansf   = __vlibm_TBL_atan1[index1] ;/* pi/2 up */
158       }
159       *y      = sign1 * ansf;           /* store answer, with sign bit  */
160       x      += stridex;
161       y      += stridey;
162       argcount = 1;                             /* we still have 1 good arg
163       if ( --n <=0 )
164       {
165         goto UNROLL;                            /* finish up with 1 good arg
166       }
167       goto LOOP1;                               /* otherwise, examine next arg
168     }
169
170     if (intf > 0x42800000)                      /* if(|x| > 64
171     {
172     f1 = -pone/f1;
173       index1 = 2;                               /* point to pi/2 upper, lower
174     }
175     else if( intf >= 0x3C800000 )               /* if |x| >= (1/64)...
176     {
177       intz   = (intf + 0x00040000) & 0x7ff80000;/* round arg, keep upper
178       pz[0]  = intz;                            /* store as a float (z)
179     f1 = (f1 - z)/(pone + f1*z);
180       index1 = (intz - 0x3C800000) >> 18;       /* (index >> 19) << 1)
181       index1 = index1 + 4;                      /* skip over 0,0,pi/2,pi/2
182     }
183     else
184     {
185         index1   = 0;                           /* points to 0,0 in table
186     }

188     yaddr1   = y;                               /* address to store this answer
189     x       += stridex;                         /* point to next arg
190     y       += stridey;                         /* point to next result
191     argcount = 2;                               /* we now have 2 good arguments
192     if ( --n <=0 )
193     {
```



```
194       goto UNROLL;                              /* finish up with 2 good args
195     }

197     /*--------------------------------------------------------------------------
198     /*--------------------------------------------------------------------------
199     /*--------------------------------------------------------------------------

201   LOOP2:

203         intf     = *(int *) x;          /* upper half of x, as integer */
204         f2 = *x;
205         sign2 = pone;
206         if (intf < 0) {
207                 intf = intf & ~0x80000000; /* abs(upper argument) */
208                 f2 = -f2;
209                 sign2 = -sign2;
210         }
211
212     if( (intf > 0x5B000000) || (intf < 0x31800000) ) /* filter out special cases
213     {
214       if( intf > 0x7f800000 )
215       {
216         ansf   = f2 - f2;                       /* return NaN if x=NaN*/
217       }
218       else if( intf < 0x31800000 )              /* avoid underflow for small arg
219       {
220         dummy = 1.0e37 + f2;
221         dummy = dummy;
222         ansf   = f2;
223       }
224       else if( intf > 0x5B000000 )              /* avoid underflow for big arg
225       {
226         index2 = 2;
227         ansf   = __vlibm_TBL_atan1[index2] ;/* pi/2 up */
228       }
229       *y      = sign2 * ansf;           /* store answer, with sign bit  */
230       x      += stridex;
231       y      += stridey;
232       argcount = 2;                             /* we still have 2 good args
233       if ( --n <=0 )
234       {
235         goto UNROLL;                            /* finish up with 2 good args
236       }
237       goto LOOP2;                               /* otherwise, examine next arg
238     }
239
240     if (intf > 0x42800000)                      /* if(|x| > 64
241     {
242     f2 = -pone/f2;
243       index2 = 2;                               /* point to pi/2 upper, lower
244     }
245     else if( intf >= 0x3C800000 )               /* if |x| >= (1/64)...
246     {
247       intz   = (intf + 0x00040000) & 0x7ff80000;/* round arg, keep upper
248       pz[0]  = intz;                            /* store as a float (z)
249     f2 = (f2 - z)/(pone + f2*z);
250       index2 = (intz - 0x3C800000) >> 18;       /* (index >> 19) << 1)
251       index2 = index2 + 4;                      /* skip over 0,0,pi/2,pi/2
252     }
253     else
254     {
255         index2   = 0;                           /* points to 0,0 in table
256     }
257     yaddr2   = y;                               /* address to store this answer
258     x       += stridex;                         /* point to next arg
259     y       += stridey;                         /* point to next result
```

```
260     argcount = 3;                           /* we now have 3 good arguments
261     if ( --n <=0 )
262     {
263       goto UNROLL;                          /* finish up with 2 good args
264     }


267     /*--------------------------------------------------------------------------
268     /*--------------------------------------------------------------------------
269     /*--------------------------------------------------------------------------

271 #ifdef UNROLL4
272   LOOP3:

274         intf     = *(int *) x;          /* upper half of x, as integer */
275         f3 = *x;
276         sign3 = pone;
277         if (intf < 0) {
278                 intf = intf & ~0x80000000; /* abs(upper argument) */
279                 f3 = -f3;
280                 sign3 = -sign3;
281         }
282
283     if( (intf > 0x5B000000) || (intf < 0x31800000) ) /* filter out special cases
284     {
285       if( intf > 0x7f800000 )
286       {
287         ansf   = f3 - f3;                   /* return NaN if x=NaN*/
288       }
289       else if( intf < 0x31800000 )          /* avoid underflow for small arg
290       {
291         dummy = 1.0e37 + f3;
292         dummy = dummy;
293         ansf   = f3;
294       }
295       else if( intf > 0x5B000000 )          /* avoid underflow for big arg
296       {
297         index3 = 2;
298         ansf   = __vlibm_TBL_atan1[index3] ;/* pi/2 up */
299       }
300       *y      = sign3 * ansf;           /* store answer, with sign bit  */
301       x      += stridex;
302       y      += stridey;
303       argcount = 3;                         /* we still have 3 good args
304       if ( --n <=0 )
305       {
306         goto UNROLL;                        /* finish up with 3 good args
307       }
308       goto LOOP3;                           /* otherwise, examine next arg
309     }
310
311     if (intf > 0x42800000)                  /* if(|x| > 64
312     {
313         n3 = -pone;
314         d3 = f3;
315     f3 = n3/d3;
316       index3 = 2;                           /* point to pi/2 upper, lower
317     }
318     else if( intf >= 0x3C800000 )           /* if |x| >= (1/64)...
319     {
320       intz   = (intf + 0x00040000) & 0x7ff80000;/* round arg, keep upper
321       pz[0]  = intz;                        /* store as a float (z)
322         n3     = (f3 - z);
323         d3     = (pone + f3*z);             /* get reduced argument
324     f3 = n3/d3;
325       index3 = (intz - 0x3C800000) >> 18;   /* (index >> 19) << 1)
```



```
326       index3 = index3 + 4;                  /* skip over 0,0,pi/2,pi/2
327     }
328     else
329     {
330         n3 = f3;
331         d3 = pone;
332         index3   = 0;                       /* points to 0,0 in table
333     }
334     yaddr3   = y;                           /* address to store this answer
335     x       += stridex;                     /* point to next arg
336     y       += stridey;                     /* point to next result
337     argcount = 4;                           /* we now have 4 good arguments
338     if ( --n <=0 )
339     {
340       goto UNROLL;                          /* finish up with 3 good args
341     }
342 #endif /* UNROLL4 */

344 /* here is the n-way unrolled section,
345    but we may actually have less than n
346    arguments at this point
347 */

349 UNROLL:

351 #ifdef UNROLL4
352     if (argcount == 4)
353     {
354     conup0   = __vlibm_TBL_atan1[index0];
355     conup1   = __vlibm_TBL_atan1[index1];
356     conup2   = __vlibm_TBL_atan1[index2];
357     conup3   = __vlibm_TBL_atan1[index3];
358     poly0    = p1*f0*f0*f0 + f0;
359     ans0     = sign0 * (float)(conup0 + poly0);
360     poly1    = p1*f1*f1*f1 + f1;
361     ans1     = sign1 * (float)(conup1 + poly1);
362     poly2    = p1*f2*f2*f2 + f2;
363     ans2     = sign2 * (float)(conup2 + poly2);
364     poly3    = p1*f3*f3*f3 + f3;
365     ans3     = sign3 * (float)(conup3 + poly3);
366     *yaddr0  = ans0;
367     *yaddr1  = ans1;
368     *yaddr2  = ans2;
369     *yaddr3  = ans3;
370     }
371     else
372 #endif
373     if (argcount == 3)
374     {
375     conup0   = __vlibm_TBL_atan1[index0];
376     conup1   = __vlibm_TBL_atan1[index1];
377     conup2   = __vlibm_TBL_atan1[index2];
378     poly0    = p1*f0*f0*f0 + f0;
379     poly1    = p1*f1*f1*f1 + f1;
380     poly2    = p1*f2*f2*f2 + f2;
381     ans0     = sign0 * (float)(conup0 + poly0);
382     ans1     = sign1 * (float)(conup1 + poly1);
383     ans2     = sign2 * (float)(conup2 + poly2);
384     *yaddr0  = ans0;
385     *yaddr1  = ans1;
386     *yaddr2  = ans2;
387     }
388     else
389     if (argcount == 2)
390     {
391     conup0   = __vlibm_TBL_atan1[index0];
```

```
 392     conup1   = __vlibm_TBL_atan1[index1];
 393     poly0    = p1*f0*f0*f0 + f0;
 394     poly1    = p1*f1*f1*f1 + f1;
 395     ans0     = sign0 * (float)(conup0 + poly0);
 396     ans1     = sign1 * (float)(conup1 + poly1);
 397     *yaddr0  = ans0;
 398     *yaddr1  = ans1;
 399     }
 400     else
 401     if (argcount == 1)
 402     {
 403     conup0   = __vlibm_TBL_atan1[index0];
 404     poly0    = p1*f0*f0*f0 + f0;
 405     ans0     = sign0 * (float)(conup0 + poly0);
 406     *yaddr0  = ans0;
 407      }

 409   }  while (n > 0);

 411 }
```

unchanged_portion_omitted_

```
**********************************************************
   29704 Sun May 11 12:16:32 2014
new/usr/src/lib/libmvec/common/__vcos.c
**********************************************************
```

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

#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 = 0.63661977236758134307553 5,  /* 53 bits of pi/2 */
        pio2_1  = 1.57079632673412561416 6,      /* first 33 bits of pi/2 */
        pio2_2  = 6.07710050630396597659 6e-11, /* second 33 bits of pi/2 */
        pio2_3  = 2.02226624871116645579 6e-21, /* third 33 bits of pi/2 */
        pio2_3t = 8.47842766036889964395 9e-32, /* pi/2 - pio2_3 */
        pp1             = -1.66666666660576046527626394313498255467 6e-0001,
        pp2             =  8.33326120969096312671837656614618094444 2e-0003,
        qq1             = -4.99999999997771098640702395590871155787 0e-0001,
        qq2             =  4.16665486385721935064505588101884208958 0e-0002,
        poly1[2]= { -1.66666666666662966980521513892030158965 6e-0001,
                                -4.99999999999993170146406087888829452448 1e-0001
        poly2[2]= {  8.33333333239095129568399345528033637666 3e-0003,
                                4.16666666394861917535640593963708222319 e-0002
        poly3[2]= { -1.98412623799797669279155177823009840396 0e-0004,
                                -1.38888855265614286783275668736851681462 e-0003
        poly4[2]= {  2.75340362485427723764998762284833035111 0e-0006,
                                2.47851942368146079661812828945453052475 9e-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;
        double          x0, x1, x2, *py0, *py1, *py2, *xsave, *ysave;
        unsigned        hx0, hx1, hx2, xsb0, xsb1, xsb2;
        int             i, biguns, nsave, sxsave, sysave;

        nsave = n;
        xsave = x;
        sxsave = stridex;
        ysave = y;
        sysave = stridey;
        biguns = 0;

        do /* MAIN LOOP */
        {
                /* Gotos here so _break_ exits MAIN LOOP. */
LOOP0:  /* Find first arg in right range. */
                xsb0 = HI(x); /* get most significant word */
                hx0 = xsb0 & ~0x80000000; /* mask off sign bit */
                if ( hx0 > 0x3fe921fb ) {
                        /* Too big: arg reduction needed, so leave for second pa
                        biguns = 1;
                        goto MEDIUM;
                }
                if ( hx0 < 0x3e400000 ) {
                        /* Too small.  cos x ~ 1. */
```

```
 55                         volatile int v = *x;
126                         *y = 1.0;
127                         x += stridex;
128                         y += stridey;
129                         i = 0;
130                         if ( --n <= 0 )
131                                 break;
132                         goto LOOP0;
133                 }
134                 x0 = *x;
135                 py0 = y;
136                 x += stridex;
137                 y += stridey;
138                 i = 1;
139                 if ( --n <= 0 )
140                         break;

142 LOOP1: /* Get second arg, same as above. */
143                 xsb1 = HI(x);
144                 hx1 = xsb1 & ~0x80000000;
145                 if ( hx1 > 0x3fe921fb )
146                 {
147                         biguns = 2;
148                         goto MEDIUM;
149                 }
150                 if ( hx1 < 0x3e400000 )
151                 {
 82                         volatile int v = *x;
152                         *y = 1.0;
153                         x += stridex;
154                         y += stridey;
155                         i = 1;
156                         if ( --n <= 0 )
157                                 break;
158                         goto LOOP1;
159                 }
160                 x1 = *x;
161                 py1 = y;
162                 x += stridex;
163                 y += stridey;
164                 i = 2;
165                 if ( --n <= 0 )
166                         break;

168 LOOP2: /* Get third arg, same as above. */
169                 xsb2 = HI(x);
170                 hx2 = xsb2 & ~0x80000000;
171                 if ( hx2 > 0x3fe921fb )
172                 {
173                         biguns = 3;
174                         goto MEDIUM;
175                 }
176                 if ( hx2 < 0x3e400000 )
177                 {
109                         volatile int v = *x;
178                         *y = 1.0;
179                         x += stridex;
180                         y += stridey;
181                         i = 2;
182                         if ( --n <= 0 )
183                                 break;
184                         goto LOOP2;
185                 }
186                 x2 = *x;
187                 py2 = y;
```



```
189                 /*
190                  * 0x3fc40000 = 5/32 ~ 0.15625
191                  * Get msb after subtraction.  Will be 1 only if
192                  * hx0 - 5/32 is negative.
193                  */
194                 i = ( hx0 - 0x3fc40000 ) >> 31;
195                 i |= ( ( hx1 - 0x3fc40000 ) >> 30 ) & 2;
196                 i |= ( ( hx2 - 0x3fc40000 ) >> 29 ) & 4;
197                 switch ( i )
198                 {
199                         double          a0, a1, a2, w0, w1, w2;
200                         double          t0, t1, t2, z0, z1, z2;
201                         unsigned        j0, j1, j2;

203                 case 0: /* All are > 5/32 */
204                         j0 = ( xsb0 + 0x4000 ) & 0xffff8000;
205                         j1 = ( xsb1 + 0x4000 ) & 0xffff8000;
206                         j2 = ( xsb2 + 0x4000 ) & 0xffff8000;
207                         HI(&t0) = j0;
208                         HI(&t1) = j1;
209                         HI(&t2) = j2;
210                         LO(&t0) = 0;
211                         LO(&t1) = 0;
212                         LO(&t2) = 0;
213                         x0 -= t0;
214                         x1 -= t1;
215                         x2 -= t2;
216                         z0 = x0 * x0;
217                         z1 = x1 * x1;
218                         z2 = x2 * x2;
219                         t0 = z0 * ( qq1 + z0 * qq2 );
220                         t1 = z1 * ( qq1 + z1 * qq2 );
221                         t2 = z2 * ( qq1 + z2 * qq2 );
222                         w0 = x0 * ( one + z0 * ( pp1 + z0 * pp2 ) );
223                         w1 = x1 * ( one + z1 * ( pp1 + z1 * pp2 ) );
224                         w2 = x2 * ( one + z2 * ( pp1 + z2 * pp2 ) );
225                         j0 = ( ( ( j0 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
226                         j1 = ( ( ( j1 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
227                         j2 = ( ( ( j2 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
228                         xsb0 = ( xsb0 >> 30 ) & 2;
229                         xsb1 = ( xsb1 >> 30 ) & 2;
230                         xsb2 = ( xsb2 >> 30 ) & 2;
231                         a0 = __vlibm_TBL_sincos_hi[j0+1]; /* cos_hi(t) */
232                         a1 = __vlibm_TBL_sincos_hi[j1+1];
233                         a2 = __vlibm_TBL_sincos_hi[j2+1];
234                            /*   cos_lo(t)                          sin_hi(t) */
235                         t0 = __vlibm_TBL_sincos_lo[j0+1] - ( __vlibm_TBL_sincos_
236                         t1 = __vlibm_TBL_sincos_lo[j1+1] - ( __vlibm_TBL_sincos_
237                         t2 = __vlibm_TBL_sincos_lo[j2+1] - ( __vlibm_TBL_sincos_

239                         *py0 = a0 + t0;
240                         *py1 = a1 + t1;
241                         *py2 = a2 + t2;
242                         break;

244                 case 1:
245                         j1 = ( xsb1 + 0x4000 ) & 0xffff8000;
246                         j2 = ( xsb2 + 0x4000 ) & 0xffff8000;
247                         HI(&t1) = j1;
248                         HI(&t2) = j2;
249                         LO(&t1) = 0;
250                         LO(&t2) = 0;
251                         x1 -= t1;
252                         x2 -= t2;
253                         z0 = x0 * x0;
254                         z1 = x1 * x1;
```

```
255                         z2 = x2 * x2;
256                         t0 = z0 * ( poly3[1] + z0 * poly4[1] );
257                         t1 = z1 * ( qq1 + z1 * qq2 );
258                         t2 = z2 * ( qq1 + z2 * qq2 );
259                         t0 = z0 * ( poly1[1] + z0 * ( poly2[1] + t0 ) );
260                         w1 = x1 * ( one + z1 * ( pp1 + z1 * pp2 ) );
261                         w2 = x2 * ( one + z2 * ( pp1 + z2 * pp2 ) );
262                         j1 = ( ( ( j1 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
263                         j2 = ( ( ( j2 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
264                         xsb1 = ( xsb1 >> 30 ) & 2;
265                         xsb2 = ( xsb2 >> 30 ) & 2;
266                         a1 = __vlibm_TBL_sincos_hi[j1+1];
267                         a2 = __vlibm_TBL_sincos_hi[j2+1];
268                         t1 = __vlibm_TBL_sincos_lo[j1+1] - ( __vlibm_TBL_sincos_
269                         t2 = __vlibm_TBL_sincos_lo[j2+1] - ( __vlibm_TBL_sincos_
270                         *py0 = one + t0;
271                         *py1 = a1 + t1;
272                         *py2 = a2 + t2;
273                         break;

275                 case 2:
276                         j0 = ( xsb0 + 0x4000 ) & 0xffff8000;
277                         j2 = ( xsb2 + 0x4000 ) & 0xffff8000;
278                         HI(&t0) = j0;
279                         HI(&t2) = j2;
280                         LO(&t0) = 0;
281                         LO(&t2) = 0;
282                         x0 -= t0;
283                         x2 -= t2;
284                         z0 = x0 * x0;
285                         z1 = x1 * x1;
286                         z2 = x2 * x2;
287                         t0 = z0 * ( qq1 + z0 * qq2 );
288                         t1 = z1 * ( poly3[1] + z1 * poly4[1] );
289                         t2 = z2 * ( qq1 + z2 * qq2 );
290                         w0 = x0 * ( one + z0 * ( pp1 + z0 * pp2 ) );
291                         t1 = z1 * ( poly1[1] + z1 * ( poly2[1] + t1 ) );
292                         w2 = x2 * ( one + z2 * ( pp1 + z2 * pp2 ) );
293                         j0 = ( ( ( j0 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
294                         j2 = ( ( ( j2 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
295                         xsb0 = ( xsb0 >> 30 ) & 2;
296                         xsb2 = ( xsb2 >> 30 ) & 2;
297                         a0 = __vlibm_TBL_sincos_hi[j0+1];
298                         a2 = __vlibm_TBL_sincos_hi[j2+1];
299                         t0 = __vlibm_TBL_sincos_lo[j0+1] - ( __vlibm_TBL_sincos_
300                         t2 = __vlibm_TBL_sincos_lo[j2+1] - ( __vlibm_TBL_sincos_
301                         *py0 = a0 + t0;
302                         *py1 = one + t1;
303                         *py2 = a2 + t2;
304                         break;

306                 case 3:
307                         j2 = ( xsb2 + 0x4000 ) & 0xffff8000;
308                         HI(&t2) = j2;
309                         LO(&t2) = 0;
310                         x2 -= t2;
311                         z0 = x0 * x0;
312                         z1 = x1 * x1;
313                         z2 = x2 * x2;
314                         t0 = z0 * ( poly3[1] + z0 * poly4[1] );
315                         t1 = z1 * ( poly3[1] + z1 * poly4[1] );
316                         t2 = z2 * ( qq1 + z2 * qq2 );
317                         t0 = z0 * ( poly1[1] + z0 * ( poly2[1] + t0 ) );
318                         t1 = z1 * ( poly1[1] + z1 * ( poly2[1] + t1 ) );
319                         w2 = x2 * ( one + z2 * ( pp1 + z2 * pp2 ) );
320                         j2 = ( ( ( j2 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
```



```
321                         xsb2 = ( xsb2 >> 30 ) & 2;
322                         a2 = __vlibm_TBL_sincos_hi[j2+1];
323                         t2 = __vlibm_TBL_sincos_lo[j2+1] - ( __vlibm_TBL_sincos_
324                         *py0 = one + t0;
325                         *py1 = one + t1;
326                         *py2 = a2 + t2;
327                         break;

329                 case 4:
330                         j0 = ( xsb0 + 0x4000 ) & 0xffff8000;
331                         j1 = ( xsb1 + 0x4000 ) & 0xffff8000;
332                         HI(&t0) = j0;
333                         HI(&t1) = j1;
334                         LO(&t0) = 0;
335                         LO(&t1) = 0;
336                         x0 -= t0;
337                         x1 -= t1;
338                         z0 = x0 * x0;
339                         z1 = x1 * x1;
340                         z2 = x2 * x2;
341                         t0 = z0 * ( qq1 + z0 * qq2 );
342                         t1 = z1 * ( qq1 + z1 * qq2 );
343                         t2 = z2 * ( poly3[1] + z2 * poly4[1] );
344                         w0 = x0 * ( one + z0 * ( pp1 + z0 * pp2 ) );
345                         w1 = x1 * ( one + z1 * ( pp1 + z1 * pp2 ) );
346                         t2 = z2 * ( poly1[1] + z2 * ( poly2[1] + t2 ) );
347                         j0 = ( ( ( j0 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
348                         j1 = ( ( ( j1 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
349                         xsb0 = ( xsb0 >> 30 ) & 2;
350                         xsb1 = ( xsb1 >> 30 ) & 2;
351                         a0 = __vlibm_TBL_sincos_hi[j0+1];
352                         a1 = __vlibm_TBL_sincos_hi[j1+1];
353                         t0 = __vlibm_TBL_sincos_lo[j0+1] - ( __vlibm_TBL_sincos_
354                         t1 = __vlibm_TBL_sincos_lo[j1+1] - ( __vlibm_TBL_sincos_
355                         *py0 = a0 + t0;
356                         *py1 = a1 + t1;
357                         *py2 = one + t2;
358                         break;

360                 case 5:
361                         j1 = ( xsb1 + 0x4000 ) & 0xffff8000;
362                         HI(&t1) = j1;
363                         LO(&t1) = 0;
364                         x1 -= t1;
365                         z0 = x0 * x0;
366                         z1 = x1 * x1;
367                         z2 = x2 * x2;
368                         t0 = z0 * ( poly3[1] + z0 * poly4[1] );
369                         t1 = z1 * ( qq1 + z1 * qq2 );
370                         t2 = z2 * ( poly3[1] + z2 * poly4[1] );
371                         t0 = z0 * ( poly1[1] + z0 * ( poly2[1] + t0 ) );
372                         w1 = x1 * ( one + z1 * ( pp1 + z1 * pp2 ) );
373                         t2 = z2 * ( poly1[1] + z2 * ( poly2[1] + t2 ) );
374                         j1 = ( ( ( j1 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
375                         xsb1 = ( xsb1 >> 30 ) & 2;
376                         a1 = __vlibm_TBL_sincos_hi[j1+1];
377                         t1 = __vlibm_TBL_sincos_lo[j1+1] - ( __vlibm_TBL_sincos_
378                         *py0 = one + t0;
379                         *py1 = a1 + t1;
380                         *py2 = one + t2;
381                         break;

383                 case 6:
384                         j0 = ( xsb0 + 0x4000 ) & 0xffff8000;
385                         HI(&t0) = j0;
386                         LO(&t0) = 0;
```

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

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

422                 x += stridex;
423                 y += stridey;
424                 i = 0;
425         } while ( --n > 0 ); /* END MAIN LOOP */

427         /*
428          * CLEAN UP last 0, 1, or 2 elts.
429          */
430         if ( i > 0 ) /* Clean up elts at tail.  i < 3. */
431         {
432                 double          a0, a1, w0, w1;
433                 double          t0, t1, z0, z1;
434                 unsigned        j0, j1;

436                 if ( i > 1 )
437                 {
438                         if ( hx1 < 0x3fc40000 )
439                         {
440                                 z1 = x1 * x1;
441                                 t1 = z1 * ( poly3[1] + z1 * poly4[1] );
442                                 t1 = z1 * ( poly1[1] + z1 * ( poly2[1] + t1 ) );
443                                 t1 = one + t1;
444                                 *py1 = t1;
445                         }
446                         else
447                         {
448                                 j1 = ( xsb1 + 0x4000 ) & 0xffff8000;
449                                 HI(&t1) = j1;
450                                 LO(&t1) = 0;
451                                 x1 -= t1;
452                                 z1 = x1 * x1;
```



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

488         return;

490         /*
491          * Take care of BIGUNS.
492          *
493          * We have jumped here in the middle of processing after having
494          * encountered a medium range argument.  Therefore things are in a
495          * bit of a tizzy.
496          */

498 MEDIUM:

500         x0_or_one[1] = 1.0;
501         x1_or_one[1] = 1.0;
502         x2_or_one[1] = 1.0;
503         x0_or_one[3] = -1.0;
504         x1_or_one[3] = -1.0;
505         x2_or_one[3] = -1.0;
506         y0_or_zero[1] = 0.0;
507         y1_or_zero[1] = 0.0;
508         y2_or_zero[1] = 0.0;
509         y0_or_zero[3] = 0.0;
510         y1_or_zero[3] = 0.0;
511         y2_or_zero[3] = 0.0;

513         if ( biguns == 3 )
514         {
515                 biguns = 0;
516                 xsb0 = xsb0 >> 31;
517                 xsb1 = xsb1 >> 31;
518                 goto loop2;
```

```
519         }
520         else if ( biguns == 2 )
521         {
522                 xsb0 = xsb0 >> 31;
523                 biguns = 0;
524                 goto loop1;
525         }
526         biguns = 0;

528         do
529         {
530                 double          fn0, fn1, fn2, a0, a1, a2, w0, w1, w2, y0, y1, y
531                 unsigned        hx;
532                 int                     n0, n1, n2;

534                 /*
535                  * Find 3 more to work on: Not already done, not too big.
536                  */

538 loop0:
539                 hx = HI(x);
540                 xsb0 = hx >> 31;
541                 hx &= ~0x80000000;
542                 if ( hx > 0x413921fb ) /* (1.6471e+06) Too big: leave it. */
543                 {
544                         if ( hx >= 0x7ff00000 ) /* Inf or NaN */
545                         {
546                                 x0 = *x;
547                                 *y = x0 - x0;
548                         }
549                         else
550                                 biguns = 1;
551                         x += stridex;
552                         y += stridey;
553                         i = 0;
554                         if ( --n <= 0 )
555                                 break;
556                         goto loop0;
557                 }
558                 x0 = *x;
559                 py0 = y;
560                 x += stridex;
561                 y += stridey;
562                 i = 1;
563                 if ( --n <= 0 )
564                         break;

566 loop1:
567                 hx = HI(x);
568                 xsb1 = hx >> 31;
569                 hx &= ~0x80000000;
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 > 0x413921fb )
599                 {
600                         if ( hx >= 0x7ff00000 )
601                         {
602                                 x2 = *x;
603                                 *y = x2 - x2;
604                         }
605                         else
606                                 biguns = 1;
607                         x += stridex;
608                         y += stridey;
609                         i = 2;
610                         if ( --n <= 0 )
611                                 break;
612                         goto loop2;
613                 }
614                 x2 = *x;
615                 py2 = y;

617                 n0 = (int) ( x0 * invpio2 + half[xsb0] );
618                 n1 = (int) ( x1 * invpio2 + half[xsb1] );
619                 n2 = (int) ( x2 * invpio2 + half[xsb2] );
620                 fn0 = (double) n0;
621                 fn1 = (double) n1;
622                 fn2 = (double) n2;
623                 n0 = (n0 + 1) & 3; /* Add 1 (before the mod) to make sin into co
624                 n1 = (n1 + 1) & 3;
625                 n2 = (n2 + 1) & 3;
626                 a0 = x0 - fn0 * pio2_1;
627                 a1 = x1 - fn1 * pio2_1;
628                 a2 = x2 - fn2 * pio2_1;
629                 w0 = fn0 * pio2_2;
630                 w1 = fn1 * pio2_2;
631                 w2 = fn2 * pio2_2;
632                 x0 = a0 - w0;
633                 x1 = a1 - w1;
634                 x2 = a2 - w2;
635                 y0 = ( a0 - x0 ) - w0;
636                 y1 = ( a1 - x1 ) - w1;
637                 y2 = ( a2 - x2 ) - w2;
638                 a0 = x0;
639                 a1 = x1;
640                 a2 = x2;
641                 w0 = fn0 * pio2_3 - y0;
642                 w1 = fn1 * pio2_3 - y1;
643                 w2 = fn2 * pio2_3 - y2;
644                 x0 = a0 - w0;
645                 x1 = a1 - w1;
646                 x2 = a2 - w2;
647                 y0 = ( a0 - x0 ) - w0;
648                 y1 = ( a1 - x1 ) - w1;
649                 y2 = ( a2 - x2 ) - w2;
650                 a0 = x0;
```

```
651                 a1 = x1;
652                 a2 = x2;
653                 w0 = fn0 * pio2_3t - y0;
654                 w1 = fn1 * pio2_3t - y1;
655                 w2 = fn2 * pio2_3t - y2;
656                 x0 = a0 - w0;
657                 x1 = a1 - w1;
658                 x2 = a2 - w2;
659                 y0 = ( a0 - x0 ) - w0;
660                 y1 = ( a1 - x1 ) - w1;
661                 y2 = ( a2 - x2 ) - w2;
662                 xsb0 = HI(&x0);
663                 i = ( ( xsb0 & ~0x80000000 ) - thresh[n0&1] ) >> 31;
664                 xsb1 = HI(&x1);
665                 i |= ( ( ( xsb1 & ~0x80000000 ) - thresh[n1&1] ) >> 30 ) & 2;
666                 xsb2 = HI(&x2);
667                 i |= ( ( ( xsb2 & ~0x80000000 ) - thresh[n2&1] ) >> 29 ) & 4;
668                 switch ( i )
669                 {
670                         double          t0, t1, t2, z0, z1, z2;
671                         unsigned        j0, j1, j2;

673                 case 0:
674                         j0 = ( xsb0 + 0x4000 ) & 0xffff8000;
675                         j1 = ( xsb1 + 0x4000 ) & 0xffff8000;
676                         j2 = ( xsb2 + 0x4000 ) & 0xffff8000;
677                         HI(&t0) = j0;
678                         HI(&t1) = j1;
679                         HI(&t2) = j2;
680                         LO(&t0) = 0;
681                         LO(&t1) = 0;
682                         LO(&t2) = 0;
683                         x0 = ( x0 - t0 ) + y0;
684                         x1 = ( x1 - t1 ) + y1;
685                         x2 = ( x2 - t2 ) + y2;
686                         z0 = x0 * x0;
687                         z1 = x1 * x1;
688                         z2 = x2 * x2;
689                         t0 = z0 * ( qq1 + z0 * qq2 );
690                         t1 = z1 * ( qq1 + z1 * qq2 );
691                         t2 = z2 * ( qq1 + z2 * qq2 );
692                         w0 = x0 * ( one + z0 * ( pp1 + z0 * pp2 ) );
693                         w1 = x1 * ( one + z1 * ( pp1 + z1 * pp2 ) );
694                         w2 = x2 * ( one + z2 * ( pp1 + z2 * pp2 ) );
695                         j0 = ( ( ( j0 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
696                         j1 = ( ( ( j1 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
697                         j2 = ( ( ( j2 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
698                         xsb0 = ( xsb0 >> 30 ) & 2;
699                         xsb1 = ( xsb1 >> 30 ) & 2;
700                         xsb2 = ( xsb2 >> 30 ) & 2;
701                         n0 ^= ( xsb0 & ~( n0 << 1 ) );
702                         n1 ^= ( xsb1 & ~( n1 << 1 ) );
703                         n2 ^= ( xsb2 & ~( n2 << 1 ) );
704                         xsb0 |= 1;
705                         xsb1 |= 1;
706                         xsb2 |= 1;
707                         a0 = __vlibm_TBL_sincos_hi[j0+n0];
708                         a1 = __vlibm_TBL_sincos_hi[j1+n1];
709                         a2 = __vlibm_TBL_sincos_hi[j2+n2];
710                         t0 = ( __vlibm_TBL_sincos_hi[j0+((n0+xsb0)&3)] * w0 + a0
711                         t1 = ( __vlibm_TBL_sincos_hi[j1+((n1+xsb1)&3)] * w1 + a1
712                         t2 = ( __vlibm_TBL_sincos_hi[j2+((n2+xsb2)&3)] * w2 + a2
713                         *py0 = ( a0 + t0 );
714                         *py1 = ( a1 + t1 );
715                         *py2 = ( a2 + t2 );
716                         break;
```



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

759                 case 2:
760                         j0 = ( xsb0 + 0x4000 ) & 0xffff8000;
761                         j1 = n1 & 1;
762                         j2 = ( xsb2 + 0x4000 ) & 0xffff8000;
763                         HI(&t0) = j0;
764                         HI(&t2) = j2;
765                         LO(&t0) = 0;
766                         LO(&t2) = 0;
767                         x1_or_one[0] = x1;
768                         x1_or_one[2] = -x1;
769                         x0 = ( x0 - t0 ) + y0;
770                         y1_or_zero[0] = y1;
771                         y1_or_zero[2] = -y1;
772                         x2 = ( x2 - t2 ) + y2;
773                         z0 = x0 * x0;
774                         z1 = x1 * x1;
775                         z2 = x2 * x2;
776                         t0 = z0 * ( qq1 + z0 * qq2 );
777                         t1 = z1 * ( poly3[j1] + z1 * poly4[j1] );
778                         t2 = z2 * ( qq1 + z2 * qq2 );
779                         w0 = x0 * ( one + z0 * ( pp1 + z0 * pp2 ) );
780                         t1 = z1 * ( poly1[j1] + z1 * ( poly2[j1] + t1 ) );
781                         w2 = x2 * ( one + z2 * ( pp1 + z2 * pp2 ) );
782                         j0 = ( ( ( j0 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
```

```
783                         j2 = ( ( ( j2 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
784                         xsb0 = ( xsb0 >> 30 ) & 2;
785                         xsb2 = ( xsb2 >> 30 ) & 2;
786                         n0 ^= ( xsb0 & ~( n0 << 1 ) );
787                         n2 ^= ( xsb2 & ~( n2 << 1 ) );
788                         xsb0 |= 1;
789                         xsb2 |= 1;
790                         a0 = __vlibm_TBL_sincos_hi[j0+n0];
791                         a2 = __vlibm_TBL_sincos_hi[j2+n2];
792                         t0 = ( __vlibm_TBL_sincos_hi[j0+((n0+xsb0)&3)] * w0 + a0
793                         t1 = x1_or_one[n1] + ( y1_or_zero[n1] + x1_or_one[n1] *
794                         t2 = ( __vlibm_TBL_sincos_hi[j2+((n2+xsb2)&3)] * w2 + a2
795                         *py0 = ( a0 + t0 );
796                         *py1 = t1;
797                         *py2 = ( a2 + t2 );
798                         break;

800                 case 3:
801                         j0 = n0 & 1;
802                         j1 = n1 & 1;
803                         j2 = ( xsb2 + 0x4000 ) & 0xffff8000;
804                         HI(&t2) = j2;
805                         LO(&t2) = 0;
806                         x0_or_one[0] = x0;
807                         x0_or_one[2] = -x0;
808                         x1_or_one[0] = x1;
809                         x1_or_one[2] = -x1;
810                         y0_or_zero[0] = y0;
811                         y0_or_zero[2] = -y0;
812                         y1_or_zero[0] = y1;
813                         y1_or_zero[2] = -y1;
814                         x2 = ( x2 - t2 ) + y2;
815                         z0 = x0 * x0;
816                         z1 = x1 * x1;
817                         z2 = x2 * x2;
818                         t0 = z0 * ( poly3[j0] + z0 * poly4[j0] );
819                         t1 = z1 * ( poly3[j1] + z1 * poly4[j1] );
820                         t2 = z2 * ( qq1 + z2 * qq2 );
821                         t0 = z0 * ( poly1[j0] + z0 * ( poly2[j0] + t0 ) );
822                         t1 = z1 * ( poly1[j1] + z1 * ( poly2[j1] + t1 ) );
823                         w2 = x2 * ( one + z2 * ( pp1 + z2 * pp2 ) );
824                         j2 = ( ( ( j2 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
825                         xsb2 = ( xsb2 >> 30 ) & 2;
826                         n2 ^= ( xsb2 & ~( n2 << 1 ) );
827                         xsb2 |= 1;
828                         a2 = __vlibm_TBL_sincos_hi[j2+n2];
829                         t0 = x0_or_one[n0] + ( y0_or_zero[n0] + x0_or_one[n0] *
830                         t1 = x1_or_one[n1] + ( y1_or_zero[n1] + x1_or_one[n1] *
831                         t2 = ( __vlibm_TBL_sincos_hi[j2+((n2+xsb2)&3)] * w2 + a2
832                         *py0 = t0;
833                         *py1 = t1;
834                         *py2 = ( a2 + t2 );
835                         break;

837                 case 4:
838                         j0 = ( xsb0 + 0x4000 ) & 0xffff8000;
839                         j1 = ( xsb1 + 0x4000 ) & 0xffff8000;
840                         j2 = n2 & 1;
841                         HI(&t0) = j0;
842                         HI(&t1) = j1;
843                         LO(&t0) = 0;
844                         LO(&t1) = 0;
845                         x2_or_one[0] = x2;
846                         x2_or_one[2] = -x2;
847                         x0 = ( x0 - t0 ) + y0;
848                         x1 = ( x1 - t1 ) + y1;
```



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

878                 case 5:
879                         j0 = n0 & 1;
880                         j1 = ( xsb1 + 0x4000 ) & 0xffff8000;
881                         j2 = n2 & 1;
882                         HI(&t1) = j1;
883                         LO(&t1) = 0;
884                         x0_or_one[0] = x0;
885                         x0_or_one[2] = -x0;
886                         x2_or_one[0] = x2;
887                         x2_or_one[2] = -x2;
888                         y0_or_zero[0] = y0;
889                         y0_or_zero[2] = -y0;
890                         x1 = ( x1 - t1 ) + y1;
891                         y2_or_zero[0] = y2;
892                         y2_or_zero[2] = -y2;
893                         z0 = x0 * x0;
894                         z1 = x1 * x1;
895                         z2 = x2 * x2;
896                         t0 = z0 * ( poly3[j0] + z0 * poly4[j0] );
897                         t1 = z1 * ( qq1 + z1 * qq2 );
898                         t2 = z2 * ( poly3[j2] + z2 * poly4[j2] );
899                         t0 = z0 * ( poly1[j0] + z0 * ( poly2[j0] + t0 ) );
900                         w1 = x1 * ( one + z1 * ( pp1 + z1 * pp2 ) );
901                         t2 = z2 * ( poly1[j2] + z2 * ( poly2[j2] + t2 ) );
902                         j1 = ( ( ( j1 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
903                         xsb1 = ( xsb1 >> 30 ) & 2;
904                         n1 ^= ( xsb1 & ~( n1 << 1 ) );
905                         xsb1 |= 1;
906                         a1 = __vlibm_TBL_sincos_hi[j1+n1];
907                         t0 = x0_or_one[n0] + ( y0_or_zero[n0] + x0_or_one[n0] *
908                         t1 = ( __vlibm_TBL_sincos_hi[j1+((n1+xsb1)&3)] * w1 + a1
909                         t2 = x2_or_one[n2] + ( y2_or_zero[n2] + x2_or_one[n2] *
910                         *py0 = t0;
911                         *py1 = ( a1 + t1 );
912                         *py2 = t2;
913                         break;
```

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

952                 case 7:
953                         j0 = n0 & 1;
954                         j1 = n1 & 1;
955                         j2 = n2 & 1;
956                         x0_or_one[0] = x0;
957                         x0_or_one[2] = -x0;
958                         x1_or_one[0] = x1;
959                         x1_or_one[2] = -x1;
960                         x2_or_one[0] = x2;
961                         x2_or_one[2] = -x2;
962                         y0_or_zero[0] = y0;
963                         y0_or_zero[2] = -y0;
964                         y1_or_zero[0] = y1;
965                         y1_or_zero[2] = -y1;
966                         y2_or_zero[0] = y2;
967                         y2_or_zero[2] = -y2;
968                         z0 = x0 * x0;
969                         z1 = x1 * x1;
970                         z2 = x2 * x2;
971                         t0 = z0 * ( poly3[j0] + z0 * poly4[j0] );
972                         t1 = z1 * ( poly3[j1] + z1 * poly4[j1] );
973                         t2 = z2 * ( poly3[j2] + z2 * poly4[j2] );
974                         t0 = z0 * ( poly1[j0] + z0 * ( poly2[j0] + t0 ) );
975                         t1 = z1 * ( poly1[j1] + z1 * ( poly2[j1] + t1 ) );
976                         t2 = z2 * ( poly1[j2] + z2 * ( poly2[j2] + t2 ) );
977                         t0 = x0_or_one[n0] + ( y0_or_zero[n0] + x0_or_one[n0] *
978                         t1 = x1_or_one[n1] + ( y1_or_zero[n1] + x1_or_one[n1] *
979                         t2 = x2_or_one[n2] + ( y2_or_zero[n2] + x2_or_one[n2] *
980                         *py0 = t0;
```



```
981                         *py1 = t1;
982                         *py2 = t2;
983                         break;
984                 }

986                 x += stridex;
987                 y += stridey;
988                 i = 0;
989         } while ( --n > 0 );

991         if ( i > 0 )
992         {
993                 double          fn0, fn1, a0, a1, w0, w1, y0, y1;
994                 double          t0, t1, z0, z1;
995                 unsigned        j0, j1;
996                 int                     n0, n1;

998                 if ( i > 1 )
999                 {
1000                        n1 = (int) ( x1 * invpio2 + half[xsb1] );
1001                        fn1 = (double) n1;
1002                        n1 = (n1 + 1) & 3; /* Add 1 (before the mod) to make sin
1003                        a1 = x1 - fn1 * pio2_1;
1004                        w1 = fn1 * pio2_2;
1005                        x1 = a1 - w1;
1006                        y1 = ( a1 - x1 ) - w1;
1007                        a1 = x1;
1008                        w1 = fn1 * pio2_3 - y1;
1009                        x1 = a1 - w1;
1010                        y1 = ( a1 - x1 ) - w1;
1011                        a1 = x1;
1012                        w1 = fn1 * pio2_3t - y1;
1013                        x1 = a1 - w1;
1014                        y1 = ( a1 - x1 ) - w1;
1015                        xsb1 = HI(&x1);
1016                        if ( ( xsb1 & ~0x80000000 ) < thresh[n1&1] )
1017                        {
1018                                j1 = n1 & 1;
1019                                x1_or_one[0] = x1;
1020                                x1_or_one[2] = -x1;
1021                                y1_or_zero[0] = y1;
1022                                y1_or_zero[2] = -y1;
1023                                z1 = x1 * x1;
1024                                t1 = z1 * ( poly3[j1] + z1 * poly4[j1] );
1025                                t1 = z1 * ( poly1[j1] + z1 * ( poly2[j1] + t1 )
1026                                t1 = x1_or_one[n1] + ( y1_or_zero[n1] + x1_or_on
1027                                *py1 = t1;
1028                        }
1029                        else
1030                        {
1031                                j1 = ( xsb1 + 0x4000 ) & 0xffff8000;
1032                                HI(&t1) = j1;
1033                                LO(&t1) = 0;
1034                                x1 = ( x1 - t1 ) + y1;
1035                                z1 = x1 * x1;
1036                                t1 = z1 * ( qq1 + z1 * qq2 );
1037                                w1 = x1 * ( one + z1 * ( pp1 + z1 * pp2 ) );
1038                                j1 = ( ( ( j1 & ~0x80000000 ) - 0x3fc40000 ) >>
1039                                xsb1 = ( xsb1 >> 30 ) & 2;
1040                                n1 ^= ( xsb1 & ~( n1 << 1 ) );
1041                                xsb1 |= 1;
1042                                a1 = __vlibm_TBL_sincos_hi[j1+n1];
1043                                t1 = ( __vlibm_TBL_sincos_hi[j1+((n1+xsb1)&3)] *
1044                                *py1 = ( a1 + t1 );
1045                        }
1046                }
```

```
1047                 n0 = (int) ( x0 * invpio2 + half[xsb0] );
1048                 fn0 = (double) n0;
1049                 n0 = (n0 + 1) & 3; /* Add 1 (before the mod) to make sin into co
1050                 a0 = x0 - fn0 * pio2_1;
1051                 w0 = fn0 * pio2_2;
1052                 x0 = a0 - w0;
1053                 y0 = ( a0 - x0 ) - w0;
1054                 a0 = x0;
1055                 w0 = fn0 * pio2_3 - y0;
1056                 x0 = a0 - w0;
1057                 y0 = ( a0 - x0 ) - w0;
1058                 a0 = x0;
1059                 w0 = fn0 * pio2_3t - y0;
1060                 x0 = a0 - w0;
1061                 y0 = ( a0 - x0 ) - w0;
1062                 xsb0 = HI(&x0);
1063                 if ( ( xsb0 & ~0x80000000 ) < thresh[n0&1] )
1064                 {
1065                         j0 = n0 & 1;
1066                         x0_or_one[0] = x0;
1067                         x0_or_one[2] = -x0;
1068                         y0_or_zero[0] = y0;
1069                         y0_or_zero[2] = -y0;
1070                         z0 = x0 * x0;
1071                         t0 = z0 * ( poly3[j0] + z0 * poly4[j0] );
1072                         t0 = z0 * ( poly1[j0] + z0 * ( poly2[j0] + t0 ) );
1073                         t0 = x0_or_one[n0] + ( y0_or_zero[n0] + x0_or_one[n0] *
1074                         *py0 = t0;
1075                 }
1076                 else
1077                 {
1078                         j0 = ( xsb0 + 0x4000 ) & 0xffff8000;
1079                         HI(&t0) = j0;
1080                         LO(&t0) = 0;
1081                         x0 = ( x0 - t0 ) + y0;
1082                         z0 = x0 * x0;
1083                         t0 = z0 * ( qq1 + z0 * qq2 );
1084                         w0 = x0 * ( one + z0 * ( pp1 + z0 * pp2 ) );
1085                         j0 = ( ( ( j0 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
1086                         xsb0 = ( xsb0 >> 30 ) & 2;
1087                         n0 ^= ( xsb0 & ~( n0 << 1 ) );
1088                         xsb0 |= 1;
1089                         a0 = __vlibm_TBL_sincos_hi[j0+n0];
1090                         t0 = ( __vlibm_TBL_sincos_hi[j0+((n0+xsb0)&3)] * w0 + a0
1091                         *py0 = ( a0 + t0 );
1092                 }
1093         }

1095         if ( biguns )
1096                 __vlibm_vcos_big( nsave, xsave, sxsave, ysave, sysave, 0x413921f
1097 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   10431 Sun May 11 12:16:33 2014
new/usr/src/lib/libmvec/common/__vcosf.c
**********************************************************
______unchanged_portion_omitted_

  76 #define S0      C[0]
  77 #define S1      C[1]
  78 #define S2      C[2]
  79 #define one     C[3]
  80 #define mhalf   C[4]
  81 #define C0      C[5]
  82 #define C1      C[6]
  83 #define C2      C[7]
  84 #define invpio2 C[8]
  85 #define c3two51 C[9]
  86 #define pio2_1  C[10]
  87 #define pio2_t  C[11]

  89 #define PREPROCESS(N, index, label)                                     \
  90         hx = *(int *)x;                                                \
  91         ix = hx & 0x7fffffff;                                          \
  92         t = *x;                                                        \
  93         x += stridex;                                                  \
  94         if (ix <= 0x3f490fdb) { /* |x| < pi/4 */                       \
  95                 if (ix == 0) {                                         \
  96                         y[index] = one;                                \
  97                         goto label;                                    \
  98                 }                                                      \
  99                 y##N = (double)t;                                      \
 100                 n##N = 1;                                              \
 101         } else if (ix <= 0x49c90fdb) { /* |x| < 2^19*pi */             \
 102                 y##N = (double)t;                                      \
 103                 medium = 1;                                            \
 104         } else {                                                       \
 105                 if (ix >= 0x7f800000) { /* inf or nan */               \
 106                         y[index] = t / t;                              \
 107                         goto label;                                    \
 108                 }                                                      \
 109                 z##N = y##N = (double)t;                               \
 110                 hx = HI(y##N);                                         \
 111                 n##N = ((hx >> 20) & 0x7ff) - 1046;                    \
 112                 HI(z##N) = (hx & 0xfffff) | 0x41600000;                \
 113                 n##N = __vlibm_rem_pio2m(&z##N, &y##N, n##N, 1, 0) + 1; \
 114                 z##N = y##N * y##N;                                    \
 115                 if (n##N & 1) { /* compute cos y */                    \
 116                         f##N = (float)(one + z##N * (mhalf + z##N *    \
 117                             (C0 + z##N * (C1 + z##N * C2))));          \
 118                 } else { /* compute sin y */                           \
 119                         f##N = (float)(y##N + y##N * z##N * (S0 +      \
 120                             z##N * (S1 + z##N * S2)));                 \
 121                 }                                                      \
 122                 y[index] = (n##N & 2)? -f##N : f##N;                   \
 123                 goto label;                                            \
 124         }

 126 #define PROCESS(N)                                                      \
 127         if (medium) {                                                  \
 128                 z##N = y##N * invpio2 + c3two51;                       \
 129                 n##N = LO(z##N) + 1;                                   \
 130                 z##N -= c3two51;                                       \
 131                 y##N = (y##N - z##N * pio2_1) - z##N * pio2_t;         \
 132         }                                                              \
 133         z##N = y##N * y##N;                                            \
 134         if (n##N & 1) { /* compute cos y */                            \
 135                 f##N = (float)(one + z##N * (mhalf + z##N * (C0 +      \
```



```
 136                     z##N * (C1 + z##N * C2))));                        \
 137         } else { /* compute sin y */                                   \
 138                 f##N = (float)(y##N + y##N * z##N * (S0 + z##N * (S1 +  \
 139                     z##N * S2)));                                      \
 140         }                                                              \
 141         *y = (n##N & 2)? -f##N : f##N;                                 \
 142         y += stridey

 144 void
 145 __vcosf(int n, float *restrict x, int stridex, float *restrict y,
 146     int stridey)
 147 {
 148         double          y0, y1, y2, y3;
 149         double          z0, z1, z2, z3;
 150         float           f0, f1, f2, f3, t;
 151         int             n0 = 0, n1 = 0, n2 = 0, n3, hx, ix, medium;
 151         int             n0, n1, n2, n3, hx, ix, medium;

 153         y -= stridey;

 155         for (;;) {
 156 begin:
 157                 y += stridey;

 159                 if (--n < 0)
 160                         break;

 162                 medium = 0;
 163                 PREPROCESS(0, 0, begin);

 165                 if (--n < 0)
 166                         goto process1;

 168                 PREPROCESS(1, stridey, process1);

 170                 if (--n < 0)
 171                         goto process2;

 173                 PREPROCESS(2, (stridey << 1), process2);

 175                 if (--n < 0)
 176                         goto process3;

 178                 PREPROCESS(3, (stridey << 1) + stridey, process3);

 180                 if (medium) {
 181                         z0 = y0 * invpio2 + c3two51;
 182                         z1 = y1 * invpio2 + c3two51;
 183                         z2 = y2 * invpio2 + c3two51;
 184                         z3 = y3 * invpio2 + c3two51;

 186                         n0 = LO(z0) + 1;
 187                         n1 = LO(z1) + 1;
 188                         n2 = LO(z2) + 1;
 189                         n3 = LO(z3) + 1;

 191                         z0 -= c3two51;
 192                         z1 -= c3two51;
 193                         z2 -= c3two51;
 194                         z3 -= c3two51;

 196                         y0 = (y0 - z0 * pio2_1) - z0 * pio2_t;
 197                         y1 = (y1 - z1 * pio2_1) - z1 * pio2_t;
 198                         y2 = (y2 - z2 * pio2_1) - z2 * pio2_t;
 199                         y3 = (y3 - z3 * pio2_1) - z3 * pio2_t;
 200                 }
```

```
202                 z0 = y0 * y0;
203                 z1 = y1 * y1;
204                 z2 = y2 * y2;
205                 z3 = y3 * y3;

207                 hx = (n0 & 1) | ((n1 & 1) << 1) | ((n2 & 1) << 2) |
208                     ((n3 & 1) << 3);
209                 switch (hx) {
210                 case 0:
211                         f0 = (float)(y0 + y0 * z0 * (S0 + z0 * (S1 + z0 * S2)));
212                         f1 = (float)(y1 + y1 * z1 * (S0 + z1 * (S1 + z1 * S2)));
213                         f2 = (float)(y2 + y2 * z2 * (S0 + z2 * (S1 + z2 * S2)));
214                         f3 = (float)(y3 + y3 * z3 * (S0 + z3 * (S1 + z3 * S2)));
215                         break;

217                 case 1:
218                         f0 = (float)(one + z0 * (mhalf + z0 * (C0 +
219                             z0 * (C1 + z0 * C2))));
220                         f1 = (float)(y1 + y1 * z1 * (S0 + z1 * (S1 + z1 * S2)));
221                         f2 = (float)(y2 + y2 * z2 * (S0 + z2 * (S1 + z2 * S2)));
222                         f3 = (float)(y3 + y3 * z3 * (S0 + z3 * (S1 + z3 * S2)));
223                         break;

225                 case 2:
226                         f0 = (float)(y0 + y0 * z0 * (S0 + z0 * (S1 + z0 * S2)));
227                         f1 = (float)(one + z1 * (mhalf + z1 * (C0 +
228                             z1 * (C1 + z1 * C2))));
229                         f2 = (float)(y2 + y2 * z2 * (S0 + z2 * (S1 + z2 * S2)));
230                         f3 = (float)(y3 + y3 * z3 * (S0 + z3 * (S1 + z3 * S2)));
231                         break;

233                 case 3:
234                         f0 = (float)(one + z0 * (mhalf + z0 * (C0 +
235                             z0 * (C1 + z0 * C2))));
236                         f1 = (float)(one + z1 * (mhalf + z1 * (C0 +
237                             z1 * (C1 + z1 * C2))));
238                         f2 = (float)(y2 + y2 * z2 * (S0 + z2 * (S1 + z2 * S2)));
239                         f3 = (float)(y3 + y3 * z3 * (S0 + z3 * (S1 + z3 * S2)));
240                         break;

242                 case 4:
243                         f0 = (float)(y0 + y0 * z0 * (S0 + z0 * (S1 + z0 * S2)));
244                         f1 = (float)(y1 + y1 * z1 * (S0 + z1 * (S1 + z1 * S2)));
245                         f2 = (float)(one + z2 * (mhalf + z2 * (C0 +
246                             z2 * (C1 + z2 * C2))));
247                         f3 = (float)(y3 + y3 * z3 * (S0 + z3 * (S1 + z3 * S2)));
248                         break;

250                 case 5:
251                         f0 = (float)(one + z0 * (mhalf + z0 * (C0 +
252                             z0 * (C1 + z0 * C2))));
253                         f1 = (float)(y1 + y1 * z1 * (S0 + z1 * (S1 + z1 * S2)));
254                         f2 = (float)(one + z2 * (mhalf + z2 * (C0 +
255                             z2 * (C1 + z2 * C2))));
256                         f3 = (float)(y3 + y3 * z3 * (S0 + z3 * (S1 + z3 * S2)));
257                         break;

259                 case 6:
260                         f0 = (float)(y0 + y0 * z0 * (S0 + z0 * (S1 + z0 * S2)));
261                         f1 = (float)(one + z1 * (mhalf + z1 * (C0 +
262                             z1 * (C1 + z1 * C2))));
263                         f2 = (float)(one + z2 * (mhalf + z2 * (C0 +
264                             z2 * (C1 + z2 * C2))));
265                         f3 = (float)(y3 + y3 * z3 * (S0 + z3 * (S1 + z3 * S2)));
266                         break;
```



```
268                 case 7:
269                         f0 = (float)(one + z0 * (mhalf + z0 * (C0 +
270                             z0 * (C1 + z0 * C2))));
271                         f1 = (float)(one + z1 * (mhalf + z1 * (C0 +
272                             z1 * (C1 + z1 * C2))));
273                         f2 = (float)(one + z2 * (mhalf + z2 * (C0 +
274                             z2 * (C1 + z2 * C2))));
275                         f3 = (float)(y3 + y3 * z3 * (S0 + z3 * (S1 + z3 * S2)));
276                         break;

278                 case 8:
279                         f0 = (float)(y0 + y0 * z0 * (S0 + z0 * (S1 + z0 * S2)));
280                         f1 = (float)(y1 + y1 * z1 * (S0 + z1 * (S1 + z1 * S2)));
281                         f2 = (float)(y2 + y2 * z2 * (S0 + z2 * (S1 + z2 * S2)));
282                         f3 = (float)(one + z3 * (mhalf + z3 * (C0 +
283                             z3 * (C1 + z3 * C2))));
284                         break;

286                 case 9:
287                         f0 = (float)(one + z0 * (mhalf + z0 * (C0 +
288                             z0 * (C1 + z0 * C2))));
289                         f1 = (float)(y1 + y1 * z1 * (S0 + z1 * (S1 + z1 * S2)));
290                         f2 = (float)(y2 + y2 * z2 * (S0 + z2 * (S1 + z2 * S2)));
291                         f3 = (float)(one + z3 * (mhalf + z3 * (C0 +
292                             z3 * (C1 + z3 * C2))));
293                         break;

295                 case 10:
296                         f0 = (float)(y0 + y0 * z0 * (S0 + z0 * (S1 + z0 * S2)));
297                         f1 = (float)(one + z1 * (mhalf + z1 * (C0 +
298                             z1 * (C1 + z1 * C2))));
299                         f2 = (float)(y2 + y2 * z2 * (S0 + z2 * (S1 + z2 * S2)));
300                         f3 = (float)(one + z3 * (mhalf + z3 * (C0 +
301                             z3 * (C1 + z3 * C2))));
302                         break;

304                 case 11:
305                         f0 = (float)(one + z0 * (mhalf + z0 * (C0 +
306                             z0 * (C1 + z0 * C2))));
307                         f1 = (float)(one + z1 * (mhalf + z1 * (C0 +
308                             z1 * (C1 + z1 * C2))));
309                         f2 = (float)(y2 + y2 * z2 * (S0 + z2 * (S1 + z2 * S2)));
310                         f3 = (float)(one + z3 * (mhalf + z3 * (C0 +
311                             z3 * (C1 + z3 * C2))));
312                         break;

314                 case 12:
315                         f0 = (float)(y0 + y0 * z0 * (S0 + z0 * (S1 + z0 * S2)));
316                         f1 = (float)(y1 + y1 * z1 * (S0 + z1 * (S1 + z1 * S2)));
317                         f2 = (float)(one + z2 * (mhalf + z2 * (C0 +
318                             z2 * (C1 + z2 * C2))));
319                         f3 = (float)(one + z3 * (mhalf + z3 * (C0 +
320                             z3 * (C1 + z3 * C2))));
321                         break;

323                 case 13:
324                         f0 = (float)(one + z0 * (mhalf + z0 * (C0 +
325                             z0 * (C1 + z0 * C2))));
326                         f1 = (float)(y1 + y1 * z1 * (S0 + z1 * (S1 + z1 * S2)));
327                         f2 = (float)(one + z2 * (mhalf + z2 * (C0 +
328                             z2 * (C1 + z2 * C2))));
329                         f3 = (float)(one + z3 * (mhalf + z3 * (C0 +
330                             z3 * (C1 + z3 * C2))));
331                         break;
```

```
333                 case 14:
334                         f0 = (float)(y0 + y0 * z0 * (S0 + z0 * (S1 + z0 * S2)));
335                         f1 = (float)(one + z1 * (mhalf + z1 * (C0 +
336                             z1 * (C1 + z1 * C2))));
337                         f2 = (float)(one + z2 * (mhalf + z2 * (C0 +
338                             z2 * (C1 + z2 * C2))));
339                         f3 = (float)(one + z3 * (mhalf + z3 * (C0 +
340                             z3 * (C1 + z3 * C2))));
341                         break;

343                 default:
344                         f0 = (float)(one + z0 * (mhalf + z0 * (C0 +
345                             z0 * (C1 + z0 * C2))));
346                         f1 = (float)(one + z1 * (mhalf + z1 * (C0 +
347                             z1 * (C1 + z1 * C2))));
348                         f2 = (float)(one + z2 * (mhalf + z2 * (C0 +
349                             z2 * (C1 + z2 * C2))));
350                         f3 = (float)(one + z3 * (mhalf + z3 * (C0 +
351                             z3 * (C1 + z3 * C2))));
352                 }

354                 *y = (n0 & 2)? -f0 : f0;
355                 y += stridey;
356                 *y = (n1 & 2)? -f1 : f1;
357                 y += stridey;
358                 *y = (n2 & 2)? -f2 : f2;
359                 y += stridey;
360                 *y = (n3 & 2)? -f3 : f3;
361                 continue;

363 process1:
364                 PROCESS(0);
365                 continue;

367 process2:
368                 PROCESS(0);
369                 PROCESS(1);
370                 continue;

372 process3:
373                 PROCESS(0);
374                 PROCESS(1);
375                 PROCESS(2);
376         }
377 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   56146 Sun May 11 12:16:35 2014
new/usr/src/lib/libmvec/common/__vpow.c
**********************************************************
_____unchanged_portion_omitted_
 546 yisint##I = 0;  /* Y - non-integer */
 547 exp = hy >> 20; /* Y exponent */
 548 ull_y0 &= LMMANT;
 549 ull_x##I = (ull_y0 | LDONE);
 549 ull_x##I = ull_y0 | LDONE;
 550 x##I = *(double*)&ull_x##I;
 551 ull_ax##I = ((ull_x##I + LMROUND) & LMHI20);
 551 ull_ax##I = (ull_x##I + LMROUND) & LMHI20;
 552 ax##I = *(double*)&ull_ax##I;
 553 if ( hx >= 0x7ff00000 || exp >= 0x43e )         /* X=Inf,Nan or |Y|>2^63,Inf,Nan
 554 {
 555         y0 = *px;
 556         if ( hx > 0x7ff00000 || (hx == 0x7ff00000 && lx != 0) ||
 557         hy > 0x7ff00000 || (hy == 0x7ff00000 && ly != 0) )      /* |X| or |Y| =
 558                 RETURN (I, y0 + *py)
 559         if ( hy == 0x7ff00000 && (ly == 0) )                    /* |Y| = Inf */
 560         {
 561                 if ( hx == 0x3ff00000 && (lx == 0) )            /* +-1 ** +-Inf
 562                         *pz = *py - *py;
 563                 else if ( (hx < 0x3ff00000) != sy )
 564                         *pz = DZERO;
 565                 else
 566                 {
 567                         HI(pz) = hy;
 568                         LO(pz) = ly;
 569                 }
 570                 RET_SC(I)
 571         }
 572         if ( exp < 0x43e )      /* |Y| < 2^63   */
 573         {
 574                 if ( sx )       /* X = -Inf     */
 575                 {
 576                         if ( exp >= 0x434 )     /* |Y| >= 2^53  */
 577                                 yisint##I = 2;  /* Y - even     */
 578                         else
 579                         {
 580                                 if ( exp >= 0x3ff )     /* |Y| >= 1     */
 581                                 {
 582                                         if ( exp > (20 + 0x3ff) )
 583                                         {
 584                                                 i0 = ly >> (52 - (exp - 0x3ff));
 585                                                 if ( (i0 << (52 - (exp - 0x3ff))
 586                                                         yisint##I = 2 - (i0 & 1)
 587                                         }
 588                                         else if ( ly == 0 )
 589                                         {
 590                                                 i0 = hy >> (20 - (exp - 0x3ff));
 591                                                 if ( (i0 << (20 - (exp - 0x3ff))
 592                                                         yisint##I = 2 - (i0 & 1)
 593                                         }
 594                                 }
 595                         }
 596                 }
 597                 if ( sy )
 598                         hx = lx = 0;
 599                 hx += yisint##I << 31;
 600                 HI(pz) = hx;
 601                 LO(pz) = lx;
 602                 RET_SC(I)
 603         }
 604         else    /* |Y| >= 2^63   */
```



```
 605         {
 606                 /* |X| = 0, 1, Inf */
 607                 if ( lx == 0 && (hx == 0 || hx == 0x3ff00000 || hx == 0x7ff00000
 608                 {
 609                         HI(pz) = hx;
 610                         LO(pz) = lx;
 611                         if ( sy )
 612                                 *pz = DONE / *pz;
 613                 }
 614                 else
 615                 {
 616                         y0 = ( (hx < 0x3ff00000) != sy ) ? _TINY : _HUGE;
 617                         *pz = y0 * y0;
 618                 }
 619                 RET_SC(I)
 620         }
 621 }
 622 if ( (sx || (hx | lx)) == 0 )   /* X <= 0       */
 622 if ( sx || (hx | lx) == 0 )     /* X <= 0       */
 623 {
 624         if ( exp >= 0x434 )     /* |Y| >= 2^53 */
 625                 yisint##I = 2;  /* Y - even    */
 626         else
 627         {
 628                 if ( exp >= 0x3ff )     /* |Y| >= 1    */
 629                 {
 630                         if ( exp > (20 + 0x3ff) )
 631                         {
 632                                 i0 = ly >> (52 - (exp - 0x3ff));
 633                                 if ( (i0 << (52 - (exp - 0x3ff))) == ly )
 634                                         yisint##I = 2 - (i0 & 1);
 635                         }
 636                         else if ( ly == 0 )
 637                         {
 638                                 i0 = hy >> (20 - (exp - 0x3ff));
 639                                 if ( (i0 << (20 - (exp - 0x3ff))) == hy )
 640                                         yisint##I = 2 - (i0 & 1);
 641                         }
 642                 }
 643         }
 644         if ( (hx | lx) == 0 )           /* X == 0  */
 645         {
 646                 y0 = DZERO;
 647                 if ( sy )
 648                         y0 = DONE / y0;
 649                 if ( sx & yisint##I )
 650                         y0 = -y0;
 651                 RETURN (I, y0)
 652         }
 653         if ( yisint##I == 0 )                   /* pow(neg,non-integer) */
 654                 RETURN (I, DZERO / DZERO)       /* NaN */
 655 }
 656 exp = (hx >> 20);
 657 exp##I = exp - 2046;
 658 py##I = py;
 659 pz##I = pz;
 660 ux##I = x##I + ax##I;
 661 if ( !exp )
 662 {
 663         ax##I = (double) ull_y0;
 664         ull_ax##I = *(unsigned long long*)&ax##I;
 665         ull_x##I = ((ull_ax##I & LMMANT) | LDONE);
 665         ull_x##I = ull_ax##I & LMMANT | LDONE;
 666         x##I = *(double*)&ull_x##I;
 667         exp##I = ((unsigned int*) & ull_ax##I)[0];
 668         exp##I = (exp##I >> 20) - (2046 + 1023 + 51);
```

```
        ull_ax##I = (ull_x##I + (LMROUND & LMHI20));
        ull_ax##I = ull_x##I + LMROUND & LMHI20;
        ax##I = *(double*)&ull_ax##I;
        ux##I = x##I + ax##I;
}
ull_x##I = *(unsigned long long *)&ux##I;
hx##I = HI(&ull_ax##I);
yd##I = DONE / ux##I;

void
__vpow( int n, double * restrict px, int stridex, double * restrict py,
        int stridey, double * restrict pz, int stridez )
{
        double                  *py0 = 0, *py1 = 0, *py2;
        double                  *pz0 = 0, *pz1 = 0, *pz2;
        double                  y0, yd0 = 0.0L, u0, s0, s_l0, m_h0;
        double                  y1, yd1 = 0.0L, u1, s1, s_l1, m_h1;
        double                  *py0, *py1, *py2;
        double                  *pz0, *pz1, *pz2;
        double                  y0, yd0, u0, s0, s_l0, m_h0;
        double                  y1, yd1, u1, s1, s_l1, m_h1;
        double                  y2, yd2, u2, s2, s_l2, m_h2;
        double                  ax0 = 0.0L, x0 = 0.0L, s_h0, ux0;
        double                  ax1 = 0.0L, x1 = 0.0L, s_h1, ux1;
        double                  ax0, x0, s_h0, ux0;
        double                  ax1, x1, s_h1, ux1;
        double                  ax2, x2, s_h2, ux2;
        int                     eflag0, gflag0, ind0, i0;
        int                     eflag1, gflag1, ind1, i1;
        int                     eflag2, gflag2, ind2, i2;
        int                     hx0 = 0, yisint0 = 0, exp0 = 0;
        int                     hx1 = 0, yisint1 = 0, exp1 = 0;
        int                     hx0, yisint0, exp0;
        int                     hx1, yisint1, exp1;
        int                     hx2, yisint2, exp2;
        int                     exp, i = 0;
        unsigned                hx, lx, sx, hy, ly, sy;
        unsigned long long      ull_y0, ull_x0, ull_x1, ull_x2, ull_ax0, ull_ax1
        unsigned long long      LDONE = ((unsigned long long*)LCONST)[1];
        unsigned long long      LMMANT = ((unsigned long long*)LCONST)[4];
        unsigned long long      LMROUND = ((unsigned long long*)LCONST)[5];
        unsigned long long      LMHI20 = ((unsigned long long*)LCONST)[6];
        double                  DONE = ((double*)LCONST)[1];
        double                  DZERO = ((double*)LCONST)[7];
        double                  KA5 = ((double*)LCONST)[8];
        double                  KA3 = ((double*)LCONST)[9];
        double                  KA1_LO = ((double*)LCONST)[10];
        double                  KA1_HI = ((double*)LCONST)[11];
        double                  KA1 = ((double*)LCONST)[12];
        double                  HTHRESH = ((double*)LCONST)[13];
        double                  LTHRESH = ((double*)LCONST)[14];
        double                  KB5 = ((double*)LCONST)[15];
        double                  KB4 = ((double*)LCONST)[16];
        double                  KB3 = ((double*)LCONST)[17];
        double                  KB2 = ((double*)LCONST)[18];
        double                  KB1 = ((double*)LCONST)[19];

        if (stridex == 0)
        {
                unsigned        hx = HI(px);
                unsigned        lx = LO(px);

                /* if x is a positive normal number not equal to one,
                   call __vpowx */
                if (hx >= 0x00100000 && hx < 0x7ff00000 &&
                        (hx != 0x3ff00000 || lx != 0))
```



```
                {
                        __vpowx( n, px, py, stridey, pz, stridez );
                        return;
                }
        }

        do
        {
                /* perform si + ydi = 256*log2(xi)*yi */
start0:
                PREP(0)
                px += stridex;
                py += stridey;
                pz += stridez;
                i = 1;
                if ( --n <= 0 )
                        break;

start1:
                PREP(1)
                px += stridex;
                py += stridey;
                pz += stridez;
                i = 2;
                if ( --n <= 0 )
                        break;

start2:
                PREP(2)

                u0 = x0 - ax0;
                u1 = x1 - ax1;
                u2 = x2 - ax2;

                s0 = u0 * yd0;
                LO(&ux0) = 0;
                s1 = u1 * yd1;
                LO(&ux1) = 0;
                s2 = u2 * yd2;
                LO(&ux2) = 0;

                y0 = s0 * s0;
                s_h0 = s0;
                LO(&s_h0) = 0;
                y1 = s1 * s1;
                s_h1 = s1;
                LO(&s_h1) = 0;
                y2 = s2 * s2;
                s_h2 = s2;
                LO(&s_h2) = 0;

                s0 = (KA5 * y0 + KA3) * y0 * s0;
                s1 = (KA5 * y1 + KA3) * y1 * s1;
                s2 = (KA5 * y2 + KA3) * y2 * s2;

                s_l0 = (x0 - (ux0 - ax0));
                s_l1 = (x1 - (ux1 - ax1));
                s_l2 = (x2 - (ux2 - ax2));

                s_l0 = u0 - s_h0 * ux0 - s_h0 * s_l0;
                s_l1 = u1 - s_h1 * ux1 - s_h1 * s_l1;
                s_l2 = u2 - s_h2 * ux2 - s_h2 * s_l2;

                s_l0 = KA1 * yd0 * s_l0;
                i0 = (hx0 >> 8) & 0xff0;
                exp0 += (hx0 >> 20);
```

```
793                 s_l1 = KA1 * yd1 * s_l1;
794                 i1 = (hx1 >> 8) & 0xff0;
795                 exp1 += (hx1 >> 20);

797                 s_l2 = KA1 * yd2 * s_l2;
798                 i2 = (hx2 >> 8) & 0xff0;
799                 exp2 += (hx2 >> 20);

801                 yd0 = KA1_HI * s_h0;
802                 yd1 = KA1_HI * s_h1;
803                 yd2 = KA1_HI * s_h2;

805                 y0 = *(double *)((char*)__TBL_log2 + i0);
806                 y1 = *(double *)((char*)__TBL_log2 + i1);
807                 y2 = *(double *)((char*)__TBL_log2 + i2);

809                 y0 += (double)(exp0 << 8);
810                 y1 += (double)(exp1 << 8);
811                 y2 += (double)(exp2 << 8);

813                 m_h0 = y0 + yd0;
814                 m_h1 = y1 + yd1;
815                 m_h2 = y2 + yd2;

817                 y0 = s0 - ((m_h0 - y0 - yd0) - s_l0);
818                 y1 = s1 - ((m_h1 - y1 - yd1) - s_l1);
819                 y2 = s2 - ((m_h2 - y2 - yd2) - s_l2);

821                 y0 += *(double *)((char*)__TBL_log2 + i0 + 8) + KA1_LO * s_h0;
822                 y1 += *(double *)((char*)__TBL_log2 + i1 + 8) + KA1_LO * s_h1;
823                 y2 += *(double *)((char*)__TBL_log2 + i2 + 8) + KA1_LO * s_h2;

825                 s_h0 = y0 + m_h0;
826                 s_h1 = y1 + m_h1;
827                 s_h2 = y2 + m_h2;

829                 LO(&s_h0) = 0;
830                 LO(&s_h1) = 0;
831                 LO(&s_h2) = 0;

833                 yd0 = *py0;
834                 yd1 = *py1;
835                 yd2 = *py2;
836                 s0 = yd0;
837                 s1 = yd1;
838                 s2 = yd2;
839                 LO(&s0) = 0;
840                 LO(&s1) = 0;
841                 LO(&s2) = 0;

843                 y0 = y0 - (s_h0 - m_h0);
844                 y1 = y1 - (s_h1 - m_h1);
845                 y2 = y2 - (s_h2 - m_h2);

847                 yd0 = (yd0 - s0) * s_h0 + yd0 * y0;
848                 yd1 = (yd1 - s1) * s_h1 + yd1 * y1;
849                 yd2 = (yd2 - s2) * s_h2 + yd2 * y2;

851                 s0 = s_h0 * s0;
852                 s1 = s_h1 * s1;
853                 s2 = s_h2 * s2;

855                 /* perform 2 ** ((si+ydi)/256) */
856                 if ( s0 > HTHRESH )
857                 {
```



```
858                         s0 = HTHRESH;
859                         yd0 = DZERO;
860                 }
861                 if ( s1 > HTHRESH )
862                 {
863                         s1 = HTHRESH;
864                         yd1 = DZERO;
865                 }
866                 if ( s2 > HTHRESH )
867                 {
868                         s2 = HTHRESH;
869                         yd2 = DZERO;
870                 }

872                 if ( s0 < LTHRESH )
873                 {
874                         s0 = LTHRESH;
875                         yd0 = DZERO;
876                 }
877                 ind0 = (int) (s0 + yd0);
878                 if ( s1 < LTHRESH )
879                 {
880                         s1 = LTHRESH;
881                         yd1 = DZERO;
882                 }
883                 ind1 = (int) (s1 + yd1);
884                 if ( s2 < LTHRESH )
885                 {
886                         s2 = LTHRESH;
887                         yd2 = DZERO;
888                 }
889                 ind2 = (int) (s2 + yd2);

891                 i0 = (ind0 & 0xff) << 4;
892                 u0 = (double) ind0;
893                 ind0 >>= 8;

895                 i1 = (ind1 & 0xff) << 4;
896                 u1 = (double)ind1;
897                 ind1 >>= 8;

899                 i2 = (ind2 & 0xff) << 4;
900                 u2 = (double) ind2;
901                 ind2 >>= 8;

903                 y0 = s0 - u0 + yd0;
904                 y1 = s1 - u1 + yd1;
905                 y2 = s2 - u2 + yd2;

907                 u0 = *(double*)((char*)__TBL_exp2 + i0);
908                 y0 = ((((KB5 * y0 + KB4) * y0 + KB3) * y0 + KB2) * y0 + KB1) * y
909                 u1 = *(double*)((char*)__TBL_exp2 + i1);
910                 y1 = ((((KB5 * y1 + KB4) * y1 + KB3) * y1 + KB2) * y1 + KB1) * y
911                 u2 = *(double*)((char*)__TBL_exp2 + i2);
912                 y2 = ((((KB5 * y2 + KB4) * y2 + KB3) * y2 + KB2) * y2 + KB1) * y

914                 eflag0 = (ind0 + 1021) >> 31;
915                 gflag0 = (1022 - ind0) >> 31;
916                 eflag1 = (ind1 + 1021) >> 31;
917                 gflag1 = (1022 - ind1) >> 31;
918                 eflag2 = (ind2 + 1021) >> 31;
919                 gflag2 = (1022 - ind2) >> 31;

921                 ind0 = (yisint0 << 11) + ind0 + (54 & eflag0) - (52 & gflag0);
922                 ind0 <<= 20;
923                 ind1 = (yisint1 << 11) + ind1 + (54 & eflag1) - (52 & gflag1);
```

```
924                 ind1 <<= 20;
925                 ind2 = (yisint2 << 11) + ind2 + (54 & eflag2) - (52 & gflag2);
926                 ind2 <<= 20;

928                 u0 = *(double*)((char*)__TBL_exp2 + i0 + 8) + u0 * y0 + u0;
929                 u1 = *(double*)((char*)__TBL_exp2 + i1 + 8) + u1 * y1 + u1;
930                 u2 = *(double*)((char*)__TBL_exp2 + i2 + 8) + u2 * y2 + u2;

932                 ull_x0 = *(unsigned long long*)&u0;
933                 HI(&ull_x0) += ind0;
934                 u0 = *(double*)&ull_x0;

936                 ull_x1 = *(unsigned long long*)&u1;
937                 HI(&ull_x1) += ind1;
938                 u1 = *(double*)&ull_x1;

940                 ull_x2 = *(unsigned long long*)&u2;
941                 HI(&ull_x2) += ind2;
942                 u2 = *(double*)&ull_x2;

944                 *pz0 = u0 * SCALE_ARR[eflag0 - gflag0];
945                 *pz1 = u1 * SCALE_ARR[eflag1 - gflag1];
946                 *pz2 = u2 * SCALE_ARR[eflag2 - gflag2];

948                 px += stridex;
949                 py += stridey;
950                 pz += stridez;
951                 i = 0;

953         } while ( --n > 0 );

955         if ( i > 0 )
956         {
957                 /* perform si + ydi = 256*log2(xi)*yi */
958                 u0 = x0 - ax0;
959                 s0 = u0 * yd0;
960                 LO(&ux0) = 0;
961                 y0 = s0 * s0;
962                 s_h0 = s0;
963                 LO(&s_h0) = 0;
964                 s0 = (KA5 * y0 + KA3) * y0 * s0;
965                 s_l0 = (x0 - (ux0 - ax0));
966                 s_l0 = u0 - s_h0 * ux0 - s_h0 * s_l0;
967                 s_l0 = KA1 * yd0 * s_l0;
968                 i0 = (hx0 >> 8) & 0xff0;
969                 exp0 += (hx0 >> 20);
970                 yd0 = KA1_HI * s_h0;
971                 y0 = *(double *)((char*)__TBL_log2 + i0);
972                 y0 += (double)(exp0 << 8);
973                 m_h0 = y0 + yd0;
974                 y0 = s0 - ((m_h0 - y0 - yd0) - s_l0);
975                 y0 += *(double *)((char*)__TBL_log2 + i0 + 8) + KA1_LO * s_h0;
976                 s_h0 = y0 + m_h0;
977                 LO(&s_h0) = 0;
978                 y0 = y0 - (s_h0 - m_h0);
979                 s0 = yd0 = *py0;
980                 LO(&s0) = 0;
981                 yd0 = (yd0 - s0) * s_h0 + yd0 * y0;
982                 s0 = s_h0 * s0;

984                 /* perform 2 ** ((si+ydi)/256) */
985                 if ( s0 > HTHRESH )
986                 {
987                         s0 = HTHRESH;
988                         yd0 = DZERO;
989                 }
```



```
990                 if ( s0 < LTHRESH )
991                 {
992                         s0 = LTHRESH;
993                         yd0 = DZERO;
994                 }
995                 ind0 = (int) (s0 + yd0);
996                 i0 = (ind0 & 0xff) << 4;
997                 u0 = (double) ind0;
998                 ind0 >>= 8;
999                 y0 = s0 - u0 + yd0;
1000                u0 = *(double*)((char*)__TBL_exp2 + i0);
1001                y0 = ((((KB5 * y0 + KB4) * y0 + KB3) * y0 + KB2) * y0 + KB1) * y
1002                eflag0 = (ind0 + 1021) >> 31;
1003                gflag0 = (1022 - ind0) >> 31;
1004                u0 = *(double*)((char*)__TBL_exp2 + i0 + 8) + u0 * y0 + u0;
1005                ind0 = (yisint0 << 11) + ind0 + (54 & eflag0) - (52 & gflag0);
1006                ind0 <<= 20;
1007                ull_x0 = *(unsigned long long*)&u0;
1008                HI(&ull_x0) += ind0;
1009                u0 = *(double*)&ull_x0;

1011                *pz0 = u0 * SCALE_ARR[eflag0 - gflag0];

1013                if ( i > 1 )
1014                {
1015                        /* perform si + ydi = 256*log2(xi)*yi */
1016                        u0 = x1 - ax1;
1017                        s0 = u0 * yd1;
1018                        LO(&ux1) = 0;
1019                        y0 = s0 * s0;
1020                        s_h0 = s0;
1021                        LO(&s_h0) = 0;
1022                        s0 = (KA5 * y0 + KA3) * y0 * s0;
1023                        s_l0 = (x1 - (ux1 - ax1));
1024                        s_l0 = u0 - s_h0 * ux1 - s_h0 * s_l0;
1025                        s_l0 = KA1 * yd1 * s_l0;
1026                        i0 = (hx1 >> 8) & 0xff0;
1027                        exp1 += (hx1 >> 20);
1028                        yd0 = KA1_HI * s_h0;
1029                        y0 = *(double *)((char*)__TBL_log2 + i0);
1030                        y0 += (double)(exp1 << 8);
1031                        m_h0 = y0 + yd0;
1032                        y0 = s0 - ((m_h0 - y0 - yd0) - s_l0);
1033                        y0 += *(double *)((char*)__TBL_log2 + i0 + 8) + KA1_LO *
1034                        s_h0 = y0 + m_h0;
1035                        LO(&s_h0) = 0;
1036                        y0 = y0 - (s_h0 - m_h0);
1037                        s0 = yd0 = *py1;
1038                        LO(&s0) = 0;
1039                        yd0 = (yd0 - s0) * s_h0 + yd0 * y0;
1040                        s0 = s_h0 * s0;
1041                        /* perform 2 ** ((si+ydi)/256) */
1042                        if ( s0 > HTHRESH )
1043                        {
1044                                s0 = HTHRESH;
1045                                yd0 = DZERO;
1046                        }
1047                        if ( s0 < LTHRESH )
1048                        {
1049                                s0 = LTHRESH;
1050                                yd0 = DZERO;
1051                        }
1052                        ind0 = (int) (s0 + yd0);
1053                        i0 = (ind0 & 0xff) << 4;
1054                        u0 = (double) ind0;
1055                        ind0 >>= 8;
```

```
1056                        y0 = s0 - u0 + yd0;
1057                        u0 = *(double*)((char*)__TBL_exp2 + i0);
1058                        y0 = ((((KB5 * y0 + KB4) * y0 + KB3) * y0 + KB2) * y0 +
1059                        eflag0 = (ind0 + 1021) >> 31;
1060                        gflag0 = (1022 - ind0) >> 31;
1061                        u0 = *(double*)((char*)__TBL_exp2 + i0 + 8) + u0 * y0 +
1062                        ind0 = (yisint1 << 11) + ind0 + (54 & eflag0) - (52 & gf
1063                        ind0 <<= 20;
1064                        ull_x0 = *(unsigned long long*)&u0;
1065                        HI(&ull_x0) += ind0;
1066                        u0 = *(double*)&ull_x0;
1067                        *pz1 = u0 * SCALE_ARR[eflag0 - gflag0];
1068                }
1069        }
1070 }
______unchanged_portion_omitted_

1115 #define LMMANT  ((unsigned long long*)LCONST)[4]        /* 0x000fffffffffffff
1116 #define LMROUND ((unsigned long long*)LCONST)[5]        /* 0x0000080000000000
1117 #define LMHI20  ((unsigned long long*)LCONST)[6]        /* 0xfffff00000000000
1118 #define MMANT   ((double*)LCONST)[4]                    /* 0x000fffffffffffff
1119 #define MROUND  ((double*)LCONST)[5]                    /* 0x0000080000000000
1120 #define MHI20   ((double*)LCONST)[6]                    /* 0xfffff00000000000
1121 #define KA5     ((double*)LCONST)[8]                    /* 5.7707860486089373798
1122 #define KA3     ((double*)LCONST)[9]                    /* 9.6179669392576554942
1123 #define KA1_LO  ((double*)LCONST)[10]                   /* 1.4105215426814730956
1124 #define KA1_HI  ((double*)LCONST)[11]                   /* 2.8853759765625e+00*2
1125 #define KA1     ((double*)LCONST)[12]                   /* 2.8853900817779267746


1128 static void
1129 __vpowx( int n, double * restrict px, double * restrict py,
1130         int stridey, double * restrict pz, int stridez )
1131 {
1132         double                  *py0, *py1 = 0, *py2;
1133         double                  *pz0, *pz1 = 0, *pz2;
1132         double                  *py0, *py1, *py2;
1133         double                  *pz0, *pz1, *pz2;
1134         double                  ux0, y0, yd0, u0, s0;
1135         double                  y1, yd1, u1, s1;
1136         double                  y2, yd2, u2, s2;
1137         double                  yr, s_h0, s_l0, m_h0, x0, ax0;
1138         unsigned long long      ull_y0, ull_x0, ull_x1, ull_x2, ull_ax0;
1139         int                     eflag0, gflag0, ind0, i0, exp0;
1140         int                     eflag1, gflag1, ind1, i1;
1141         int                     eflag2, gflag2, ind2, i2;
1142         int                     i = 0;
1143         unsigned                hx, hx0, hy, ly, sy;
1144         double                  DONE = ((double*)LCONST)[1];
1145         unsigned long long      LDONE = ((unsigned long long*)LCONST)[1];
1146         double                  DZERO = ((double*)LCONST)[7];
1147         double                  HTHRESH = ((double*)LCONST)[13];
1148         double                  LTHRESH = ((double*)LCONST)[14];
1149         double                  KB5 = ((double*)LCONST)[15];
1150         double                  KB4 = ((double*)LCONST)[16];
1151         double                  KB3 = ((double*)LCONST)[17];
1152         double                  KB2 = ((double*)LCONST)[18];
1153         double                  KB1 = ((double*)LCONST)[19];

1155         /* perform s_h + yr = 256*log2(x) */
1156         ull_y0 = *(unsigned long long*)px;
1157         hx = HI(px);
1158         ull_x0 = (ull_y0 & LMMANT) | LDONE;
1158         ull_x0 = ull_y0 & LMMANT | LDONE;
1159         x0 = *(double*)&ull_x0;
1160         exp0 = (hx >> 20) - 2046;
```



```
1161         ull_ax0 = ull_x0 + (LMROUND & LMHI20);
1161         ull_ax0 = ull_x0 + LMROUND & LMHI20;
1162         ax0 = *(double*)&ull_ax0;
1163         hx0 = HI(&ax0);
1164         ux0 = x0 + ax0;
1165         yd0 = DONE / ux0;
1166         u0 = x0 - ax0;
1167         s0 = u0 * yd0;
1168         LO(&ux0) = 0;
1169         y0 = s0 * s0;
1170         s_h0 = s0;
1171         LO(&s_h0) = 0;
1172         s0 = (KA5 * y0 + KA3) * y0 * s0;
1173         s_l0 = (x0 - (ux0 - ax0));
1174         s_l0 = u0 - s_h0 * ux0 - s_h0 * s_l0;
1175         s_l0 = KA1 * yd0 * s_l0;
1176         i0 = (hx0 >> 8) & 0xff0;
1177         exp0 += (hx0 >> 20);
1178         yd0 = KA1_HI * s_h0;
1179         y0 = *(double *)((char*)__TBL_log2 + i0);
1180         y0 += (double)(exp0 << 8);
1181         m_h0 = y0 + yd0;
1182         y0 = s0 - ((m_h0 - y0 - yd0) - s_l0);
1183         y0 += *(double *)((char*)__TBL_log2 + i0 + 8) + KA1_LO * s_h0;
1184         s_h0 = y0 + m_h0;
1185         LO(&s_h0) = 0;
1186         yr = y0 - (s_h0 - m_h0);

1188         do
1189         {
1190         /* perform 2 ** ((s_h0+yr)*yi/256) */
1191 start0:
1192                 PREP_X(0)
1193                 py += stridey;
1194                 pz += stridez;
1195                 i = 1;
1196                 if ( --n <= 0 )
1197                         break;

1199 start1:
1200                 PREP_X(1)
1201                 py += stridey;
1202                 pz += stridez;
1203                 i = 2;
1204                 if ( --n <= 0 )
1205                         break;

1207 start2:
1208                 PREP_X(2)

1210                 s0 = yd0 = *py0;
1211                 s1 = yd1 = *py1;
1212                 s2 = yd2 = *py2;

1214                 LO(&s0) = 0;
1215                 LO(&s1) = 0;
1216                 LO(&s2) = 0;

1218                 yd0 = (yd0 - s0) * s_h0 + yd0 * yr;
1219                 yd1 = (yd1 - s1) * s_h0 + yd1 * yr;
1220                 yd2 = (yd2 - s2) * s_h0 + yd2 * yr;

1222                 s0 = s_h0 * s0;
1223                 s1 = s_h0 * s1;
1224                 s2 = s_h0 * s2;
```

```
1226                if ( s0 > HTHRESH )
1227                {
1228                        s0 = HTHRESH;
1229                        yd0 = DZERO;
1230                }
1231                if ( s1 > HTHRESH )
1232                {
1233                        s1 = HTHRESH;
1234                        yd1 = DZERO;
1235                }
1236                if ( s2 > HTHRESH )
1237                {
1238                        s2 = HTHRESH;
1239                        yd2 = DZERO;
1240                }

1242                if ( s0 < LTHRESH )
1243                {
1244                        s0 = LTHRESH;
1245                        yd0 = DZERO;
1246                }
1247                ind0 = (int) (s0 + yd0);
1248                if ( s1 < LTHRESH )
1249                {
1250                        s1 = LTHRESH;
1251                        yd1 = DZERO;
1252                }
1253                ind1 = (int) (s1 + yd1);
1254                if ( s2 < LTHRESH )
1255                {
1256                        s2 = LTHRESH;
1257                        yd2 = DZERO;
1258                }
1259                ind2 = (int) (s2 + yd2);

1261                i0 = (ind0 & 0xff) << 4;
1262                u0 = (double) ind0;
1263                ind0 >>= 8;

1265                i1 = (ind1 & 0xff) << 4;
1266                u1 = (double) ind1;
1267                ind1 >>= 8;

1269                i2 = (ind2 & 0xff) << 4;
1270                u2 = (double) ind2;
1271                ind2 >>= 8;

1273                y0 = s0 - u0 + yd0;
1274                y1 = s1 - u1 + yd1;
1275                y2 = s2 - u2 + yd2;

1277                u0 = *(double*)((char*)__TBL_exp2 + i0);
1278                y0 = ((((KB5 * y0 + KB4) * y0 + KB3) * y0 + KB2) * y0 + KB1) * y
1279                u1 = *(double*)((char*)__TBL_exp2 + i1);
1280                y1 = ((((KB5 * y1 + KB4) * y1 + KB3) * y1 + KB2) * y1 + KB1) * y
1281                u2 = *(double*)((char*)__TBL_exp2 + i2);
1282                y2 = ((((KB5 * y2 + KB4) * y2 + KB3) * y2 + KB2) * y2 + KB1) * y

1284                eflag0 = (ind0 + 1021) >> 31;
1285                gflag0 = (1022 - ind0) >> 31;
1286                eflag1 = (ind1 + 1021) >> 31;
1287                gflag1 = (1022 - ind1) >> 31;
1288                eflag2 = (ind2 + 1021) >> 31;
1289                gflag2 = (1022 - ind2) >> 31;

1291                u0 = *(double*)((char*)__TBL_exp2 + i0 + 8) + u0 * y0 + u0;
```



```
1292                ind0 = ind0 + (54 & eflag0) - (52 & gflag0);
1293                ind0 <<= 20;
1294                ull_x0 = *(unsigned long long*)&u0;
1295                HI(&ull_x0) += ind0;
1296                u0 = *(double*)&ull_x0;

1298                u1 = *(double*)((char*)__TBL_exp2 + i1 + 8) + u1 * y1 + u1;
1299                ind1 = ind1 + (54 & eflag1) - (52 & gflag1);
1300                ind1 <<= 20;
1301                ull_x1 = *(unsigned long long*)&u1;
1302                HI(&ull_x1) += ind1;
1303                u1 = *(double*)&ull_x1;

1305                u2 = *(double*)((char*)__TBL_exp2 + i2 + 8) + u2 * y2 + u2;
1306                ind2 = ind2 + (54 & eflag2) - (52 & gflag2);
1307                ind2 <<= 20;
1308                ull_x2 = *(unsigned long long*)&u2;
1309                HI(&ull_x2) += ind2;
1310                u2 = *(double*)&ull_x2;

1312                *pz0 = u0 * SCALE_ARR[eflag0 - gflag0];
1313                *pz1 = u1 * SCALE_ARR[eflag1 - gflag1];
1314                *pz2 = u2 * SCALE_ARR[eflag2 - gflag2];

1316                py += stridey;
1317                pz += stridez;
1318                i = 0;

1320        } while ( --n > 0 );

1322        if ( i > 0 )
1323        {
1324                /* perform 2 ** ((s_h0+yr)*yi/256) */
1325                s0 = y0 = *py0;
1326                LO(&s0) = 0;
1327                yd0 = (y0 - s0) * s_h0 + y0 * yr;
1328                s0 = s_h0 * s0;
1329                if ( s0 > HTHRESH )
1330                {
1331                        s0 = HTHRESH;
1332                        yd0 = DZERO;
1333                }
1334                if ( s0 < LTHRESH )
1335                {
1336                        s0 = LTHRESH;
1337                        yd0 = DZERO;
1338                }
1339                ind0 = (int) (s0 + yd0);
1340                i0 = (ind0 & 0xff) << 4;
1341                u0 = (double) ind0;
1342                ind0 >>= 8;
1343                y0 = s0 - u0 + yd0;
1344                u0 = *(double*)((char*)__TBL_exp2 + i0);
1345                y0 = ((((KB5 * y0 + KB4) * y0 + KB3) * y0 + KB2) * y0 + KB1) * y
1346                eflag0 = (ind0 + 1021) >> 31;
1347                gflag0 = (1022 - ind0) >> 31;
1348                u0 = *(double*)((char*)__TBL_exp2 + i0 + 8) + u0 * y0 + u0;
1349                ind0 = ind0 + (54 & eflag0) - (52 & gflag0);
1350                ind0 <<= 20;
1351                ull_x0 = *(unsigned long long*)&u0;
1352                HI(&ull_x0) += ind0;
1353                u0 = *(double*)&ull_x0;
1354                *pz0 = u0 * SCALE_ARR[eflag0 - gflag0];

1356                if ( i > 1 )
1357                {
```

```
1358                        /* perform 2 ** ((s_h0+yr)*yi/256) */
1359                        s0 = y0 = *py1;
1360                        LO(&s0) = 0;
1361                        yd0 = (y0 - s0) * s_h0 + y0 * yr;
1362                        s0 = s_h0 * s0;
1363                        if ( s0 > HTHRESH )
1364                        {
1365                                s0 = HTHRESH;
1366                                yd0 = DZERO;
1367                        }
1368                        if ( s0 < LTHRESH )
1369                        {
1370                                s0 = LTHRESH;
1371                                yd0 = DZERO;
1372                        }
1373                        ind0 = (int) (s0 + yd0);
1374                        i0 = (ind0 & 0xff) << 4;
1375                        u0 = (double) ind0;
1376                        ind0 >>= 8;
1377                        y0 = s0 - u0 + yd0;
1378                        u0 = *(double*)((char*)__TBL_exp2 + i0);
1379                        y0 = ((((KB5 * y0 + KB4) * y0 + KB3) * y0 + KB2) * y0 +
1380                        eflag0 = (ind0 + 1021) >> 31;
1381                        gflag0 = (1022 - ind0) >> 31;
1382                        u0 = *(double*)((char*)__TBL_exp2 + i0 + 8) + u0 * y0 +
1383                        ind0 = ind0 + (54 & eflag0) - (52 & gflag0);
1384                        ind0 <<= 20;
1385                        ull_x0 = *(unsigned long long*)&u0;
1386                        HI(&ull_x0) += ind0;
1387                        u0 = *(double*)&ull_x0;
1388                        *pz1 = u0 * SCALE_ARR[eflag0 - gflag0];
1389                }
1390        }
1391 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   11672 Sun May 11 12:16:37 2014
new/usr/src/lib/libmvec/common/__vrhypot.c
**********************************************************
______unchanged_portion_omitted_
 206 j0 = hy##I - (diff0 & j0);
 207 j0 &= 0x7ff00000;
 208 HI(&scl##I) = 0x7ff00000 - j0;

 210 void
 211 __vrhypot( int n, double * restrict px, int stridex, double * restrict py,
 212         int stridey, double * restrict pz, int stridez )
 213 {
 214         int             i = 0;
 215         double          x, y;
 216         double          x_hi0, x_lo0, y_hi0, y_lo0, scl0 = 0;
 217         double          x0, y0, res0, dd0;
 218         double          res0_hi,res0_lo, dres0;
 219         double          x_hi1, x_lo1, y_hi1, y_lo1, scl1 = 0;
 220         double          x1 = 0.0L, y1 = 0.0L, res1, dd1;
 220         double          x1, y1, res1, dd1;
 221         double          res1_hi,res1_lo, dres1;
 222         double          x_hi2, x_lo2, y_hi2, y_lo2, scl2 = 0;
 223         double          x2, y2, res2, dd2;
 224         double          res2_hi,res2_lo, dres2;

 226         int             hx0, hy0, j0, diff0;
 227         int             iarr0, iexp0, itbl0;
 228         int             hx1, hy1;
 229         int             iarr1, iexp1, itbl1;
 230         int             hx2, hy2;
 231         int             iarr2, iexp2, itbl2;

 233         int             lx, ly;

 235         double          DONE = ((double*)LCONST)[0];
 236         double          DTWO = ((double*)LCONST)[1];
 237         double          D2ON36 = ((double*)LCONST)[2];
 238         double          D2ON1022 = ((double*)LCONST)[3];
 239         double          D2ONM52 = ((double*)LCONST)[4];

 241         double          *pz0, *pz1 = 0, *pz2;
 241         double          *pz0, *pz1, *pz2;

 243         do
 244         {
 245 start0:
 246                 PREP(0)
 247                 px += stridex;
 248                 py += stridey;
 249                 pz += stridez;
 250                 i = 1;
 251                 if ( --n <= 0 )
 252                         break;

 254 start1:
 255                 PREP(1)
 256                 px += stridex;
 257                 py += stridey;
 258                 pz += stridez;
 259                 i = 2;
 260                 if ( --n <= 0 )
 261                         break;

 263 start2:
 264                 PREP(2)
```



```
 266                 x0 *= scl0;
 267                 y0 *= scl0;
 268                 x1 *= scl1;
 269                 y1 *= scl1;
 270                 x2 *= scl2;
 271                 y2 *= scl2;

 273                 x_hi0 = ( x0 + D2ON36 ) - D2ON36;
 274                 y_hi0 = ( y0 + D2ON36 ) - D2ON36;
 275                 x_hi1 = ( x1 + D2ON36 ) - D2ON36;
 276                 y_hi1 = ( y1 + D2ON36 ) - D2ON36;
 277                 x_hi2 = ( x2 + D2ON36 ) - D2ON36;
 278                 y_hi2 = ( y2 + D2ON36 ) - D2ON36;
 279                 x_lo0 = x0 - x_hi0;
 280                 y_lo0 = y0 - y_hi0;
 281                 x_lo1 = x1 - x_hi1;
 282                 y_lo1 = y1 - y_hi1;
 283                 x_lo2 = x2 - x_hi2;
 284                 y_lo2 = y2 - y_hi2;
 285                 res0_hi = (x_hi0 * x_hi0 + y_hi0 * y_hi0);
 286                 res1_hi = (x_hi1 * x_hi1 + y_hi1 * y_hi1);
 287                 res2_hi = (x_hi2 * x_hi2 + y_hi2 * y_hi2);
 288                 res0_lo = ((x0 + x_hi0) * x_lo0 + (y0 + y_hi0) * y_lo0);
 289                 res1_lo = ((x1 + x_hi1) * x_lo1 + (y1 + y_hi1) * y_lo1);
 290                 res2_lo = ((x2 + x_hi2) * x_lo2 + (y2 + y_hi2) * y_lo2);

 292                 dres0 = res0_hi + res0_lo;
 293                 dres1 = res1_hi + res1_lo;
 294                 dres2 = res2_hi + res2_lo;

 296                 iarr0 = HI(&dres0);
 297                 iarr1 = HI(&dres1);
 298                 iarr2 = HI(&dres2);
 299                 iexp0 = iarr0 & 0xfff00000;
 300                 iexp1 = iarr1 & 0xfff00000;
 301                 iexp2 = iarr2 & 0xfff00000;

 303                 iarr0 = (iarr0 >> 11) & 0x1fc;
 304                 iarr1 = (iarr1 >> 11) & 0x1fc;
 305                 iarr2 = (iarr2 >> 11) & 0x1fc;
 306                 itbl0 = ((int*)((char*)__vlibm_TBL_rhypot + iarr0))[0];
 307                 itbl1 = ((int*)((char*)__vlibm_TBL_rhypot + iarr1))[0];
 308                 itbl2 = ((int*)((char*)__vlibm_TBL_rhypot + iarr2))[0];
 309                 itbl0 -= iexp0;
 310                 itbl1 -= iexp1;
 311                 itbl2 -= iexp2;
 312                 HI(&dd0) = itbl0;
 313                 HI(&dd1) = itbl1;
 314                 HI(&dd2) = itbl2;
 315                 LO(&dd0) = 0;
 316                 LO(&dd1) = 0;
 317                 LO(&dd2) = 0;

 319                 dd0 = dd0 * (DTWO - dd0 * dres0);
 320                 dd1 = dd1 * (DTWO - dd1 * dres1);
 321                 dd2 = dd2 * (DTWO - dd2 * dres2);
 322                 dd0 = dd0 * (DTWO - dd0 * dres0);
 323                 dd1 = dd1 * (DTWO - dd1 * dres1);
 324                 dd2 = dd2 * (DTWO - dd2 * dres2);
 325                 dres0 = dd0 * (DTWO - dd0 * dres0);
 326                 dres1 = dd1 * (DTWO - dd1 * dres1);
 327                 dres2 = dd2 * (DTWO - dd2 * dres2);

 329                 HI(&res0) = HI(&dres0) & 0xffffff00;
 330                 HI(&res1) = HI(&dres1) & 0xffffff00;
```

```
331                 HI(&res2) = HI(&dres2) & 0xffffff00;
332                 LO(&res0) = 0;
333                 LO(&res1) = 0;
334                 LO(&res2) = 0;
335                 res0 += (DONE - res0_hi * res0 - res0_lo * res0) * dres0;
336                 res1 += (DONE - res1_hi * res1 - res1_lo * res1) * dres1;
337                 res2 += (DONE - res2_hi * res2 - res2_lo * res2) * dres2;
338                 res0 = sqrt ( res0 );
339                 res1 = sqrt ( res1 );
340                 res2 = sqrt ( res2 );

342                 res0 = scl0 * res0;
343                 res1 = scl1 * res1;
344                 res2 = scl2 * res2;

346                 *pz0 = res0;
347                 *pz1 = res1;
348                 *pz2 = res2;

350                 px += stridex;
351                 py += stridey;
352                 pz += stridez;
353                 i = 0;

355         } while ( --n > 0 );

357         if ( i > 0 )
358         {
359                 x0 *= scl0;
360                 y0 *= scl0;

362                 x_hi0 = ( x0 + D2ON36 ) - D2ON36;
363                 y_hi0 = ( y0 + D2ON36 ) - D2ON36;
364                 x_lo0 = x0 - x_hi0;
365                 y_lo0 = y0 - y_hi0;
366                 res0_hi = (x_hi0 * x_hi0 + y_hi0 * y_hi0);
367                 res0_lo = ((x0 + x_hi0) * x_lo0 + (y0 + y_hi0) * y_lo0);

369                 dres0 = res0_hi + res0_lo;

371                 iarr0 = HI(&dres0);
372                 iexp0 = iarr0 & 0xfff00000;

374                 iarr0 = (iarr0 >> 11) & 0x1fc;
375                 itbl0 = ((int*)((char*)__vlibm_TBL_rhypot + iarr0))[0];
376                 itbl0 -= iexp0;
377                 HI(&dd0) = itbl0;
378                 LO(&dd0) = 0;

380                 dd0 = dd0 * (DTWO - dd0 * dres0);
381                 dd0 = dd0 * (DTWO - dd0 * dres0);
382                 dres0 = dd0 * (DTWO - dd0 * dres0);

384                 HI(&res0) = HI(&dres0) & 0xffffff00;
385                 LO(&res0) = 0;
386                 res0 += (DONE - res0_hi * res0 - res0_lo * res0) * dres0;
387                 res0 = sqrt ( res0 );

389                 res0 = scl0 * res0;

391                 *pz0 = res0;

393                 if ( i > 1 )
394                 {
395                         x1 *= scl1;
396                         y1 *= scl1;
```



```
398                         x_hi1 = ( x1 + D2ON36 ) - D2ON36;
399                         y_hi1 = ( y1 + D2ON36 ) - D2ON36;
400                         x_lo1 = x1 - x_hi1;
401                         y_lo1 = y1 - y_hi1;
402                         res1_hi = (x_hi1 * x_hi1 + y_hi1 * y_hi1);
403                         res1_lo = ((x1 + x_hi1) * x_lo1 + (y1 + y_hi1) * y_lo1);

405                         dres1 = res1_hi + res1_lo;

407                         iarr1 = HI(&dres1);
408                         iexp1 = iarr1 & 0xfff00000;

410                         iarr1 = (iarr1 >> 11) & 0x1fc;
411                         itbl1 = ((int*)((char*)__vlibm_TBL_rhypot + iarr1))[0];
412                         itbl1 -= iexp1;
413                         HI(&dd1) = itbl1;
414                         LO(&dd1) = 0;

416                         dd1 = dd1 * (DTWO - dd1 * dres1);
417                         dd1 = dd1 * (DTWO - dd1 * dres1);
418                         dres1 = dd1 * (DTWO - dd1 * dres1);

420                         HI(&res1) = HI(&dres1) & 0xffffff00;
421                         LO(&res1) = 0;
422                         res1 += (DONE - res1_hi * res1 - res1_lo * res1) * dres1
423                         res1 = sqrt ( res1 );

425                         res1 = scl1 * res1;

427                         *pz1 = res1;
428                 }
429         }
430 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   28751 Sun May 11 12:16:38 2014
new/usr/src/lib/libmvec/common/__vsin.c
**********************************************************
```

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

#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 = 0.6366197723675813430755350,
        pio2_1  = 1.5707963267341256141660,
        pio2_2  = 6.0771005063039659765960e-11,
        pio2_3  = 2.0222662487111664557960e-21,
        pio2_3t = 8.4784276603688996439590e-32,
        pp1             = -1.6666666666057604652762639431349825546760e-0001,
        pp2             =  8.3332612096909631267183765661461809444420e-0003,
        qq1             = -4.9999999999777109864070239559087115578700e-0001,
        qq2             =  4.1666548638572193506450558810188420895800e-0002,
        poly1[2]= { -1.6666666666666296698052151389203015896560e-0001,
```



```
                                -4.9999999999999317014640608788882945244810e-0001
        poly2[2]= {  8.3333333239095129568399345528033637666300e-0003,
                                 4.1666666663948619175356405939637082223190e-0002
        poly3[2]= { -1.9841262379979766927915517782300984039600e-0004,
                                -1.3888885526561428678327566873685168146200e-0003
        poly4[2]= {  2.7534036248542772376499876228483303511100e-0006,
                                 2.4785194236814607966181282894545305247590e-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;
        double          x0, x1, x2, *py0, *py1, *py2, *xsave, *ysave;
        unsigned        hx0, hx1, hx2, xsb0, xsb1, xsb2;
        int             i, biguns, nsave, sxsave, sysave;

        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 )
                {
                        volatile int v = *x;
                        *y = *x;
                        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:
                xsb1 = HI(x);
                hx1 = xsb1 & ~0x80000000;
                if ( hx1 > 0x3fe921fb )
```

```
125                 {
126                         biguns = 2;
127                         goto MEDIUM;
128                 }
129                 if ( hx1 < 0x3e400000 )
130                 {
 81                         volatile int v = *x;
131                         *y = *x;
132                         x += stridex;
133                         y += stridey;
134                         i = 1;
135                         if ( --n <= 0 )
136                                 break;
137                         goto LOOP1;
138                 }
139                 x1 = *x;
140                 py1 = y;
141                 x += stridex;
142                 y += stridey;
143                 i = 2;
144                 if ( --n <= 0 )
145                         break;

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

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

177                 case 0:
178                         j0 = ( xsb0 + 0x4000 ) & 0xffff8000;
179                         j1 = ( xsb1 + 0x4000 ) & 0xffff8000;
180                         j2 = ( xsb2 + 0x4000 ) & 0xffff8000;
181                         HI(&t0) = j0;
182                         HI(&t1) = j1;
183                         HI(&t2) = j2;
184                         LO(&t0) = 0;
185                         LO(&t1) = 0;
186                         LO(&t2) = 0;
187                         x0 -= t0;
188                         x1 -= t1;
```



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

216                 case 1:
217                         j1 = ( xsb1 + 0x4000 ) & 0xffff8000;
218                         j2 = ( xsb2 + 0x4000 ) & 0xffff8000;
219                         HI(&t1) = j1;
220                         HI(&t2) = j2;
221                         LO(&t1) = 0;
222                         LO(&t2) = 0;
223                         x1 -= t1;
224                         x2 -= t2;
225                         z0 = x0 * x0;
226                         z1 = x1 * x1;
227                         z2 = x2 * x2;
228                         t0 = z0 * ( poly3[0] + z0 * poly4[0] );
229                         t1 = z1 * ( qq1 + z1 * qq2 );
230                         t2 = z2 * ( qq1 + z2 * qq2 );
231                         t0 = z0 * ( poly1[0] + z0 * ( poly2[0] + t0 ) );
232                         w1 = x1 * ( one + z1 * ( pp1 + z1 * pp2 ) );
233                         w2 = x2 * ( one + z2 * ( pp1 + z2 * pp2 ) );
234                         j1 = ( ( ( j1 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
235                         j2 = ( ( ( j2 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
236                         xsb1 = ( xsb1 >> 30 ) & 2;
237                         xsb2 = ( xsb2 >> 30 ) & 2;
238                         a1 = __vlibm_TBL_sincos_hi[j1+xsb1];
239                         a2 = __vlibm_TBL_sincos_hi[j2+xsb2];
240                         t0 = x0 + x0 * t0;
241                         t1 = ( __vlibm_TBL_sincos_hi[j1+1] * w1 + a1 * t1 ) + __
242                         t2 = ( __vlibm_TBL_sincos_hi[j2+1] * w2 + a2 * t2 ) + __
243                         *py0 = t0;
244                         *py1 = a1 + t1;
245                         *py2 = a2 + t2;
246                         break;

248                 case 2:
249                         j0 = ( xsb0 + 0x4000 ) & 0xffff8000;
250                         j2 = ( xsb2 + 0x4000 ) & 0xffff8000;
251                         HI(&t0) = j0;
252                         HI(&t2) = j2;
253                         LO(&t0) = 0;
254                         LO(&t2) = 0;
```

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

280                 case 3:
281                         j2 = ( xsb2 + 0x4000 ) & 0xffff8000;
282                         HI(&t2) = j2;
283                         LO(&t2) = 0;
284                         x2 -= t2;
285                         z0 = x0 * x0;
286                         z1 = x1 * x1;
287                         z2 = x2 * x2;
288                         t0 = z0 * ( poly3[0] + z0 * poly4[0] );
289                         t1 = z1 * ( poly3[0] + z1 * poly4[0] );
290                         t2 = z2 * ( qq1 + z2 * qq2 );
291                         t0 = z0 * ( poly1[0] + z0 * ( poly2[0] + t0 ) );
292                         t1 = z1 * ( poly1[0] + z1 * ( poly2[0] + t1 ) );
293                         w2 = x2 * ( one + z2 * ( pp1 + z2 * pp2 ) );
294                         j2 = ( ( ( j2 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
295                         xsb2 = ( xsb2 >> 30 ) & 2;
296                         a2 = __vlibm_TBL_sincos_hi[j2+xsb2];
297                         t0 = x0 + x0 * t0;
298                         t1 = x1 + x1 * t1;
299                         t2 = ( __vlibm_TBL_sincos_hi[j2+1] * w2 + a2 * t2 ) + __
300                         *py0 = t0;
301                         *py1 = t1;
302                         *py2 = a2 + t2;
303                         break;

305                 case 4:
306                         j0 = ( xsb0 + 0x4000 ) & 0xffff8000;
307                         j1 = ( xsb1 + 0x4000 ) & 0xffff8000;
308                         HI(&t0) = j0;
309                         HI(&t1) = j1;
310                         LO(&t0) = 0;
311                         LO(&t1) = 0;
312                         x0 -= t0;
313                         x1 -= t1;
314                         z0 = x0 * x0;
315                         z1 = x1 * x1;
316                         z2 = x2 * x2;
317                         t0 = z0 * ( qq1 + z0 * qq2 );
318                         t1 = z1 * ( qq1 + z1 * qq2 );
319                         t2 = z2 * ( poly3[0] + z2 * poly4[0] );
320                         w0 = x0 * ( one + z0 * ( pp1 + z0 * pp2 ) );
```



```
321                         w1 = x1 * ( one + z1 * ( pp1 + z1 * pp2 ) );
322                         t2 = z2 * ( poly1[0] + z2 * ( poly2[0] + t2 ) );
323                         j0 = ( ( ( j0 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
324                         j1 = ( ( ( j1 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
325                         xsb0 = ( xsb0 >> 30 ) & 2;
326                         xsb1 = ( xsb1 >> 30 ) & 2;
327                         a0 = __vlibm_TBL_sincos_hi[j0+xsb0];
328                         a1 = __vlibm_TBL_sincos_hi[j1+xsb1];
329                         t0 = ( __vlibm_TBL_sincos_hi[j0+1] * w0 + a0 * t0 ) + __
330                         t1 = ( __vlibm_TBL_sincos_hi[j1+1] * w1 + a1 * t1 ) + __
331                         t2 = x2 + x2 * t2;
332                         *py0 = a0 + t0;
333                         *py1 = a1 + t1;
334                         *py2 = t2;
335                         break;

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

362                 case 6:
363                         j0 = ( xsb0 + 0x4000 ) & 0xffff8000;
364                         HI(&t0) = j0;
365                         LO(&t0) = 0;
366                         x0 -= t0;
367                         z0 = x0 * x0;
368                         z1 = x1 * x1;
369                         z2 = x2 * x2;
370                         t0 = z0 * ( qq1 + z0 * qq2 );
371                         t1 = z1 * ( poly3[0] + z1 * poly4[0] );
372                         t2 = z2 * ( poly3[0] + z2 * poly4[0] );
373                         w0 = x0 * ( one + z0 * ( pp1 + z0 * pp2 ) );
374                         t1 = z1 * ( poly1[0] + z1 * ( poly2[0] + t1 ) );
375                         t2 = z2 * ( poly1[0] + z2 * ( poly2[0] + t2 ) );
376                         j0 = ( ( ( j0 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
377                         xsb0 = ( xsb0 >> 30 ) & 2;
378                         a0 = __vlibm_TBL_sincos_hi[j0+xsb0];
379                         t0 = ( __vlibm_TBL_sincos_hi[j0+1] * w0 + a0 * t0 ) + __
380                         t1 = x1 + x1 * t1;
381                         t2 = x2 + x2 * t2;
382                         *py0 = a0 + t0;
383                         *py1 = t1;
384                         *py2 = t2;
385                         break;
```

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

406                 x += stridex;
407                 y += stridey;
408                 i = 0;
409         } while ( --n > 0 );

411         if ( i > 0 )
412         {
413                 double          a0, a1, w0, w1;
414                 double          t0, t1, z0, z1;
415                 unsigned        j0, j1;

417                 if ( i > 1 )
418                 {
419                         if ( hx1 < 0x3fc90000 )
420                         {
421                                 z1 = x1 * x1;
422                                 t1 = z1 * ( poly3[0] + z1 * poly4[0] );
423                                 t1 = z1 * ( poly1[0] + z1 * ( poly2[0] + t1 ) );
424                                 t1 = x1 + x1 * t1;
425                                 *py1 = t1;
426                         }
427                         else
428                         {
429                                 j1 = ( xsb1 + 0x4000 ) & 0xffff8000;
430                                 HI(&t1) = j1;
431                                 LO(&t1) = 0;
432                                 x1 -= t1;
433                                 z1 = x1 * x1;
434                                 t1 = z1 * ( qq1 + z1 * qq2 );
435                                 w1 = x1 * ( one + z1 * ( pp1 + z1 * pp2 ) );
436                                 j1 = ( ( ( j1 & ~0x80000000 ) - 0x3fc40000 ) >>
437                                 xsb1 = ( xsb1 >> 30 ) & 2;
438                                 a1 = __vlibm_TBL_sincos_hi[j1+xsb1];
439                                 t1 = ( __vlibm_TBL_sincos_hi[j1+1] * w1 + a1 * t
440                                 *py1 = a1 + t1;
441                         }
442                 }
443                 if ( hx0 < 0x3fc90000 )
444                 {
445                         z0 = x0 * x0;
446                         t0 = z0 * ( poly3[0] + z0 * poly4[0] );
447                         t0 = z0 * ( poly1[0] + z0 * ( poly2[0] + t0 ) );
448                         t0 = x0 + x0 * t0;
449                         *py0 = t0;
450                 }
451                 else
452                 {
```



```
453                         j0 = ( xsb0 + 0x4000 ) & 0xffff8000;
454                         HI(&t0) = j0;
455                         LO(&t0) = 0;
456                         x0 -= t0;
457                         z0 = x0 * x0;
458                         t0 = z0 * ( qq1 + z0 * qq2 );
459                         w0 = x0 * ( one + z0 * ( pp1 + z0 * pp2 ) );
460                         j0 = ( ( ( j0 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
461                         xsb0 = ( xsb0 >> 30 ) & 2;
462                         a0 = __vlibm_TBL_sincos_hi[j0+xsb0];
463                         t0 = ( __vlibm_TBL_sincos_hi[j0+1] * w0 + a0 * t0 ) + __
464                         *py0 = a0 + t0;
465                 }
466         }

468         return;

470         /*
471          * MEDIUM RANGE PROCESSING
472          * Jump here at first sign of medium range argument.  We are a bit
473          * confused due to the jump.. fix up several variables and jump into
474          * the nth loop, same as was being processed above.
475          */

477 MEDIUM:

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

492         if ( biguns == 3 )
493         {
494                 biguns = 0;
495                 xsb0 = xsb0 >> 31;
496                 xsb1 = xsb1 >> 31;
497                 goto loop2;
498         }
499         else if ( biguns == 2 )
500         {
501                 xsb0 = xsb0 >> 31;
502                 biguns = 0;
503                 goto loop1;
504         }
505         biguns = 0;

507         do
508         {
509                 double          fn0, fn1, fn2, a0, a1, a2, w0, w1, w2, y0, y1, y
510                 unsigned        hx;
511                 int             n0, n1, n2;

513 loop0:
514                 hx = HI(x);
515                 xsb0 = hx >> 31;
516                 hx &= ~0x80000000;
517                 if ( hx < 0x3e400000 )
518                 {
```

```
471                         volatile int v = *x;
519                         *y = *x;
520                         x += stridex;
521                         y += stridey;
522                         i = 0;
523                         if ( --n <= 0 )
524                                 break;
525                         goto loop0;
526                 }
527                 if ( hx > 0x413921fb )
528                 {
529                         if ( hx >= 0x7ff00000 )
530                         {
531                                 x0 = *x;
532                                 *y = x0 - x0;
533                         }
534                         else
535                                 biguns = 1;
536                         x += stridex;
537                         y += stridey;
538                         i = 0;
539                         if ( --n <= 0 )
540                                 break;
541                         goto loop0;
542                 }
543                 x0 = *x;
544                 py0 = y;
545                 x += stridex;
546                 y += stridey;
547                 i = 1;
548                 if ( --n <= 0 )
549                         break;

551 loop1:
552                 hx = HI(x);
553                 xsb1 = hx >> 31;
554                 hx &= ~0x80000000;
555                 if ( hx < 0x3e400000 )
556                 {
510                         volatile int v = *x;
557                         *y = *x;
558                         x += stridex;
559                         y += stridey;
560                         i = 1;
561                         if ( --n <= 0 )
562                                 break;
563                         goto loop1;
564                 }
565                 if ( hx > 0x413921fb )
566                 {
567                         if ( hx >= 0x7ff00000 )
568                         {
569                                 x1 = *x;
570                                 *y = x1 - x1;
571                         }
572                         else
573                                 biguns = 1;
574                         x += stridex;
575                         y += stridey;
576                         i = 1;
577                         if ( --n <= 0 )
578                                 break;
579                         goto loop1;
580                 }
581                 x1 = *x;
582                 py1 = y;
```



```
583                 x += stridex;
584                 y += stridey;
585                 i = 2;
586                 if ( --n <= 0 )
587                         break;

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

622                 n0 = (int) ( x0 * invpio2 + half[xsb0] );
623                 n1 = (int) ( x1 * invpio2 + half[xsb1] );
624                 n2 = (int) ( x2 * invpio2 + half[xsb2] );
625                 fn0 = (double) n0;
626                 fn1 = (double) n1;
627                 fn2 = (double) n2;
628                 n0 &= 3;
629                 n1 &= 3;
630                 n2 &= 3;
631                 a0 = x0 - fn0 * pio2_1;
632                 a1 = x1 - fn1 * pio2_1;
633                 a2 = x2 - fn2 * pio2_1;
634                 w0 = fn0 * pio2_2;
635                 w1 = fn1 * pio2_2;
636                 w2 = fn2 * pio2_2;
637                 x0 = a0 - w0;
638                 x1 = a1 - w1;
639                 x2 = a2 - w2;
640                 y0 = ( a0 - x0 ) - w0;
641                 y1 = ( a1 - x1 ) - w1;
642                 y2 = ( a2 - x2 ) - w2;
643                 a0 = x0;
644                 a1 = x1;
645                 a2 = x2;
646                 w0 = fn0 * pio2_3 - y0;
647                 w1 = fn1 * pio2_3 - y1;
```

```
                w2 = fn2 * pio2_3 - y2;
                x0 = a0 - w0;
                x1 = a1 - w1;
                x2 = a2 - w2;
                y0 = ( a0 - x0 ) - w0;
                y1 = ( a1 - x1 ) - w1;
                y2 = ( a2 - x2 ) - w2;
                a0 = x0;
                a1 = x1;
                a2 = x2;
                w0 = fn0 * pio2_3t - y0;
                w1 = fn1 * pio2_3t - y1;
                w2 = fn2 * pio2_3t - y2;
                x0 = a0 - w0;
                x1 = a1 - w1;
                x2 = a2 - w2;
                y0 = ( a0 - x0 ) - w0;
                y1 = ( a1 - x1 ) - w1;
                y2 = ( a2 - x2 ) - w2;
                xsb0 = HI(&x0);
                i = ( ( xsb0 & ~0x80000000 ) - thresh[n0&1] ) >> 31;
                xsb1 = HI(&x1);
                i |= ( ( ( xsb1 & ~0x80000000 ) - thresh[n1&1] ) >> 30 ) & 2;
                xsb2 = HI(&x2);
                i |= ( ( ( xsb2 & ~0x80000000 ) - thresh[n2&1] ) >> 29 ) & 4;
                switch ( i )
                {
                        double          t0, t1, t2, z0, z1, z2;
                        unsigned        j0, j1, j2;

                case 0:
                        j0 = ( xsb0 + 0x4000 ) & 0xffff8000;
                        j1 = ( xsb1 + 0x4000 ) & 0xffff8000;
                        j2 = ( xsb2 + 0x4000 ) & 0xffff8000;
                        HI(&t0) = j0;
                        HI(&t1) = j1;
                        HI(&t2) = j2;
                        LO(&t0) = 0;
                        LO(&t1) = 0;
                        LO(&t2) = 0;
                        x0 = ( x0 - t0 ) + y0;
                        x1 = ( x1 - t1 ) + y1;
                        x2 = ( x2 - t2 ) + y2;
                        z0 = x0 * x0;
                        z1 = x1 * x1;
                        z2 = x2 * x2;
                        t0 = z0 * ( qq1 + z0 * qq2 );
                        t1 = z1 * ( qq1 + z1 * qq2 );
                        t2 = z2 * ( qq1 + z2 * qq2 );
                        w0 = x0 * ( one + z0 * ( pp1 + z0 * pp2 ) );
                        w1 = x1 * ( one + z1 * ( pp1 + z1 * pp2 ) );
                        w2 = x2 * ( one + z2 * ( pp1 + z2 * pp2 ) );
                        j0 = ( ( ( j0 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
                        j1 = ( ( ( j1 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
                        j2 = ( ( ( j2 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
                        xsb0 = ( xsb0 >> 30 ) & 2;
                        xsb1 = ( xsb1 >> 30 ) & 2;
                        xsb2 = ( xsb2 >> 30 ) & 2;
                        n0 ^= ( xsb0 & ~( n0 << 1 ) );
                        n1 ^= ( xsb1 & ~( n1 << 1 ) );
                        n2 ^= ( xsb2 & ~( n2 << 1 ) );
                        xsb0 |= 1;
                        xsb1 |= 1;
                        xsb2 |= 1;
                        a0 = __vlibm_TBL_sincos_hi[j0+n0];
                        a1 = __vlibm_TBL_sincos_hi[j1+n1];
```



```
                        a2 = __vlibm_TBL_sincos_hi[j2+n2];
                        t0 = ( __vlibm_TBL_sincos_hi[j0+((n0+xsb0)&3)] * w0 + a0
                        t1 = ( __vlibm_TBL_sincos_hi[j1+((n1+xsb1)&3)] * w1 + a1
                        t2 = ( __vlibm_TBL_sincos_hi[j2+((n2+xsb2)&3)] * w2 + a2
                        *py0 = ( a0 + t0 );
                        *py1 = ( a1 + t1 );
                        *py2 = ( a2 + t2 );
                        break;

                case 1:
                        j0 = n0 & 1;
                        j1 = ( xsb1 + 0x4000 ) & 0xffff8000;
                        j2 = ( xsb2 + 0x4000 ) & 0xffff8000;
                        HI(&t1) = j1;
                        HI(&t2) = j2;
                        LO(&t1) = 0;
                        LO(&t2) = 0;
                        x0_or_one[0] = x0;
                        x0_or_one[2] = -x0;
                        y0_or_zero[0] = y0;
                        y0_or_zero[2] = -y0;
                        x1 = ( x1 - t1 ) + y1;
                        x2 = ( x2 - t2 ) + y2;
                        z0 = x0 * x0;
                        z1 = x1 * x1;
                        z2 = x2 * x2;
                        t0 = z0 * ( poly3[j0] + z0 * poly4[j0] );
                        t1 = z1 * ( qq1 + z1 * qq2 );
                        t2 = z2 * ( qq1 + z2 * qq2 );
                        t0 = z0 * ( poly1[j0] + z0 * ( poly2[j0] + 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;
                        n1 ^= ( xsb1 & ~( n1 << 1 ) );
                        n2 ^= ( xsb2 & ~( n2 << 1 ) );
                        xsb1 |= 1;
                        xsb2 |= 1;
                        a1 = __vlibm_TBL_sincos_hi[j1+n1];
                        a2 = __vlibm_TBL_sincos_hi[j2+n2];
                        t0 = x0_or_one[n0] + ( y0_or_zero[n0] + x0_or_one[n0] *
                        t1 = ( __vlibm_TBL_sincos_hi[j1+((n1+xsb1)&3)] * w1 + a1
                        t2 = ( __vlibm_TBL_sincos_hi[j2+((n2+xsb2)&3)] * w2 + a2
                        *py0 = t0;
                        *py1 = ( a1 + t1 );
                        *py2 = ( a2 + t2 );
                        break;

                case 2:
                        j0 = ( xsb0 + 0x4000 ) & 0xffff8000;
                        j1 = n1 & 1;
                        j2 = ( xsb2 + 0x4000 ) & 0xffff8000;
                        HI(&t0) = j0;
                        HI(&t2) = j2;
                        LO(&t0) = 0;
                        LO(&t2) = 0;
                        x1_or_one[0] = x1;
                        x1_or_one[2] = -x1;
                        x0 = ( x0 - t0 ) + y0;
                        y1_or_zero[0] = y1;
                        y1_or_zero[2] = -y1;
                        x2 = ( x2 - t2 ) + y2;
                        z0 = x0 * x0;
                        z1 = x1 * x1;
```

```
780                         z2 = x2 * x2;
781                         t0 = z0 * ( qq1 + z0 * qq2 );
782                         t1 = z1 * ( poly3[j1] + z1 * poly4[j1] );
783                         t2 = z2 * ( qq1 + z2 * qq2 );
784                         w0 = x0 * ( one + z0 * ( pp1 + z0 * pp2 ) );
785                         t1 = z1 * ( poly1[j1] + z1 * ( poly2[j1] + t1 ) );
786                         w2 = x2 * ( one + z2 * ( pp1 + z2 * pp2 ) );
787                         j0 = ( ( ( j0 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
788                         j2 = ( ( ( j2 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
789                         xsb0 = ( xsb0 >> 30 ) & 2;
790                         xsb2 = ( xsb2 >> 30 ) & 2;
791                         n0 ^= ( xsb0 & ~( n0 << 1 ) );
792                         n2 ^= ( xsb2 & ~( n2 << 1 ) );
793                         xsb0 |= 1;
794                         xsb2 |= 1;
795                         a0 = __vlibm_TBL_sincos_hi[j0+n0];
796                         a2 = __vlibm_TBL_sincos_hi[j2+n2];
797                         t0 = ( __vlibm_TBL_sincos_hi[j0+((n0+xsb0)&3)] * w0 + a0
798                         t1 = x1_or_one[n1] + ( y1_or_zero[n1] + x1_or_one[n1] *
799                         t2 = ( __vlibm_TBL_sincos_hi[j2+((n2+xsb2)&3)] * w2 + a2
800                         *py0 = ( a0 + t0 );
801                         *py1 = t1;
802                         *py2 = ( a2 + t2 );
803                         break;

805                 case 3:
806                         j0 = n0 & 1;
807                         j1 = n1 & 1;
808                         j2 = ( xsb2 + 0x4000 ) & 0xffff8000;
809                         HI(&t2) = j2;
810                         LO(&t2) = 0;
811                         x0_or_one[0] = x0;
812                         x0_or_one[2] = -x0;
813                         x1_or_one[0] = x1;
814                         x1_or_one[2] = -x1;
815                         y0_or_zero[0] = y0;
816                         y0_or_zero[2] = -y0;
817                         y1_or_zero[0] = y1;
818                         y1_or_zero[2] = -y1;
819                         x2 = ( x2 - t2 ) + y2;
820                         z0 = x0 * x0;
821                         z1 = x1 * x1;
822                         z2 = x2 * x2;
823                         t0 = z0 * ( poly3[j0] + z0 * poly4[j0] );
824                         t1 = z1 * ( poly3[j1] + z1 * poly4[j1] );
825                         t2 = z2 * ( qq1 + z2 * qq2 );
826                         t0 = z0 * ( poly1[j0] + z0 * ( poly2[j0] + t0 ) );
827                         t1 = z1 * ( poly1[j1] + z1 * ( poly2[j1] + t1 ) );
828                         w2 = x2 * ( one + z2 * ( pp1 + z2 * pp2 ) );
829                         j2 = ( ( ( j2 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
830                         xsb2 = ( xsb2 >> 30 ) & 2;
831                         n2 ^= ( xsb2 & ~( n2 << 1 ) );
832                         xsb2 |= 1;
833                         a2 = __vlibm_TBL_sincos_hi[j2+n2];
834                         t0 = x0_or_one[n0] + ( y0_or_zero[n0] + x0_or_one[n0] *
835                         t1 = x1_or_one[n1] + ( y1_or_zero[n1] + x1_or_one[n1] *
836                         t2 = ( __vlibm_TBL_sincos_hi[j2+((n2+xsb2)&3)] * w2 + a2
837                         *py0 = t0;
838                         *py1 = t1;
839                         *py2 = ( a2 + t2 );
840                         break;

842                 case 4:
843                         j0 = ( xsb0 + 0x4000 ) & 0xffff8000;
844                         j1 = ( xsb1 + 0x4000 ) & 0xffff8000;
845                         j2 = n2 & 1;
```



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

883                 case 5:
884                         j0 = n0 & 1;
885                         j1 = ( xsb1 + 0x4000 ) & 0xffff8000;
886                         j2 = n2 & 1;
887                         HI(&t1) = j1;
888                         LO(&t1) = 0;
889                         x0_or_one[0] = x0;
890                         x0_or_one[2] = -x0;
891                         x2_or_one[0] = x2;
892                         x2_or_one[2] = -x2;
893                         y0_or_zero[0] = y0;
894                         y0_or_zero[2] = -y0;
895                         x1 = ( x1 - t1 ) + y1;
896                         y2_or_zero[0] = y2;
897                         y2_or_zero[2] = -y2;
898                         z0 = x0 * x0;
899                         z1 = x1 * x1;
900                         z2 = x2 * x2;
901                         t0 = z0 * ( poly3[j0] + z0 * poly4[j0] );
902                         t1 = z1 * ( qq1 + z1 * qq2 );
903                         t2 = z2 * ( poly3[j2] + z2 * poly4[j2] );
904                         t0 = z0 * ( poly1[j0] + z0 * ( poly2[j0] + t0 ) );
905                         w1 = x1 * ( one + z1 * ( pp1 + z1 * pp2 ) );
906                         t2 = z2 * ( poly1[j2] + z2 * ( poly2[j2] + t2 ) );
907                         j1 = ( ( ( j1 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
908                         xsb1 = ( xsb1 >> 30 ) & 2;
909                         n1 ^= ( xsb1 & ~( n1 << 1 ) );
910                         xsb1 |= 1;
911                         a1 = __vlibm_TBL_sincos_hi[j1+n1];
```

```
                        t0 = x0_or_one[n0] + ( y0_or_zero[n0] + x0_or_one[n0] *
                        t1 = ( __vlibm_TBL_sincos_hi[j1+((n1+xsb1)&3)] * w1 + a1
                        t2 = x2_or_one[n2] + ( y2_or_zero[n2] + x2_or_one[n2] *
                        *py0 = t0;
                        *py1 = ( a1 + t1 );
                        *py2 = t2;
                        break;

                case 6:
                        j0 = ( xsb0 + 0x4000 ) & 0xffff8000;
                        j1 = n1 & 1;
                        j2 = n2 & 1;
                        HI(&t0) = j0;
                        LO(&t0) = 0;
                        x1_or_one[0] = x1;
                        x1_or_one[2] = -x1;
                        x2_or_one[0] = x2;
                        x2_or_one[2] = -x2;
                        x0 = ( x0 - t0 ) + y0;
                        y1_or_zero[0] = y1;
                        y1_or_zero[2] = -y1;
                        y2_or_zero[0] = y2;
                        y2_or_zero[2] = -y2;
                        z0 = x0 * x0;
                        z1 = x1 * x1;
                        z2 = x2 * x2;
                        t0 = z0 * ( qq1 + z0 * qq2 );
                        t1 = z1 * ( poly3[j1] + z1 * poly4[j1] );
                        t2 = z2 * ( poly3[j2] + z2 * poly4[j2] );
                        w0 = x0 * ( one + z0 * ( pp1 + z0 * pp2 ) );
                        t1 = z1 * ( poly1[j1] + z1 * ( poly2[j1] + t1 ) );
                        t2 = z2 * ( poly1[j2] + z2 * ( poly2[j2] + t2 ) );
                        j0 = ( ( ( j0 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
                        xsb0 = ( xsb0 >> 30 ) & 2;
                        n0 ^= ( xsb0 & ~( n0 << 1 ) );
                        xsb0 |= 1;
                        a0 = __vlibm_TBL_sincos_hi[j0+n0];
                        t0 = ( __vlibm_TBL_sincos_hi[j0+((n0+xsb0)&3)] * w0 + a0
                        t1 = x1_or_one[n1] + ( y1_or_zero[n1] + x1_or_one[n1] *
                        t2 = x2_or_one[n2] + ( y2_or_zero[n2] + x2_or_one[n2] *
                        *py0 = ( a0 + t0 );
                        *py1 = t1;
                        *py2 = t2;
                        break;

                case 7:
                        j0 = n0 & 1;
                        j1 = n1 & 1;
                        j2 = n2 & 1;
                        x0_or_one[0] = x0;
                        x0_or_one[2] = -x0;
                        x1_or_one[0] = x1;
                        x1_or_one[2] = -x1;
                        x2_or_one[0] = x2;
                        x2_or_one[2] = -x2;
                        y0_or_zero[0] = y0;
                        y0_or_zero[2] = -y0;
                        y1_or_zero[0] = y1;
                        y1_or_zero[2] = -y1;
                        y2_or_zero[0] = y2;
                        y2_or_zero[2] = -y2;
                        z0 = x0 * x0;
                        z1 = x1 * x1;
                        z2 = x2 * x2;
                        t0 = z0 * ( poly3[j0] + z0 * poly4[j0] );
                        t1 = z1 * ( poly3[j1] + z1 * poly4[j1] );
```



```
                        t2 = z2 * ( poly3[j2] + z2 * poly4[j2] );
                        t0 = z0 * ( poly1[j0] + z0 * ( poly2[j0] + t0 ) );
                        t1 = z1 * ( poly1[j1] + z1 * ( poly2[j1] + t1 ) );
                        t2 = z2 * ( poly1[j2] + z2 * ( poly2[j2] + t2 ) );
                        t0 = x0_or_one[n0] + ( y0_or_zero[n0] + x0_or_one[n0] *
                        t1 = x1_or_one[n1] + ( y1_or_zero[n1] + x1_or_one[n1] *
                        t2 = x2_or_one[n2] + ( y2_or_zero[n2] + x2_or_one[n2] *
                        *py0 = t0;
                        *py1 = t1;
                        *py2 = t2;
                        break;
                }

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

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

                if ( i > 1 )
                {
                        n1 = (int) ( x1 * invpio2 + half[xsb1] );
                        fn1 = (double) n1;
                        n1 &= 3;
                        a1 = x1 - fn1 * pio2_1;
                        w1 = fn1 * pio2_2;
                        x1 = a1 - w1;
                        y1 = ( a1 - x1 ) - w1;
                        a1 = x1;
                        w1 = fn1 * pio2_3 - y1;
                        x1 = a1 - w1;
                        y1 = ( a1 - x1 ) - w1;
                        a1 = x1;
                        w1 = fn1 * pio2_3t - y1;
                        x1 = a1 - w1;
                        y1 = ( a1 - x1 ) - w1;
                        xsb1 = HI(&x1);
                        if ( ( xsb1 & ~0x80000000 ) < thresh[n1&1] )
                        {
                                j1 = n1 & 1;
                                x1_or_one[0] = x1;
                                x1_or_one[2] = -x1;
                                y1_or_zero[0] = y1;
                                y1_or_zero[2] = -y1;
                                z1 = x1 * x1;
                                t1 = z1 * ( poly3[j1] + z1 * poly4[j1] );
                                t1 = z1 * ( poly1[j1] + z1 * ( poly2[j1] + t1 )
                                t1 = x1_or_one[n1] + ( y1_or_zero[n1] + x1_or_on
                                *py1 = t1;
                        }
                        else
                        {
                                j1 = ( xsb1 + 0x4000 ) & 0xffff8000;
                                HI(&t1) = j1;
                                LO(&t1) = 0;
                                x1 = ( x1 - t1 ) + y1;
                                z1 = x1 * x1;
                                t1 = z1 * ( qq1 + z1 * qq2 );
                                w1 = x1 * ( one + z1 * ( pp1 + z1 * pp2 ) );
                                j1 = ( ( ( j1 & ~0x80000000 ) - 0x3fc40000 ) >>
```

```
1044                                    xsb1 = ( xsb1 >> 30 ) & 2;
1045                                    n1 ^= ( xsb1 & ~( n1 << 1 ) );
1046                                    xsb1 |= 1;
1047                                    a1 = __vlibm_TBL_sincos_hi[j1+n1];
1048                                    t1 = ( __vlibm_TBL_sincos_hi[j1+((n1+xsb1)&3)] *
1049                                    *py1 = ( a1 + t1 );
1050                            }
1051                    }
1052                    n0 = (int) ( x0 * invpio2 + half[xsb0] );
1053                    fn0 = (double) n0;
1054                    n0 &= 3;
1055                    a0 = x0 - fn0 * pio2_1;
1056                    w0 = fn0 * pio2_2;
1057                    x0 = a0 - w0;
1058                    y0 = ( a0 - x0 ) - w0;
1059                    a0 = x0;
1060                    w0 = fn0 * pio2_3 - y0;
1061                    x0 = a0 - w0;
1062                    y0 = ( a0 - x0 ) - w0;
1063                    a0 = x0;
1064                    w0 = fn0 * pio2_3t - y0;
1065                    x0 = a0 - w0;
1066                    y0 = ( a0 - x0 ) - w0;
1067                    xsb0 = HI(&x0);
1068                    if ( ( xsb0 & ~0x80000000 ) < thresh[n0&1] )
1069                    {
1070                            j0 = n0 & 1;
1071                            x0_or_one[0] = x0;
1072                            x0_or_one[2] = -x0;
1073                            y0_or_zero[0] = y0;
1074                            y0_or_zero[2] = -y0;
1075                            z0 = x0 * x0;
1076                            t0 = z0 * ( poly3[j0] + z0 * poly4[j0] );
1077                            t0 = z0 * ( poly1[j0] + z0 * ( poly2[j0] + t0 ) );
1078                            t0 = x0_or_one[n0] + ( y0_or_zero[n0] + x0_or_one[n0] *
1079                            *py0 = t0;
1080                    }
1081                    else
1082                    {
1083                            j0 = ( xsb0 + 0x4000 ) & 0xffff8000;
1084                            HI(&t0) = j0;
1085                            LO(&t0) = 0;
1086                            x0 = ( x0 - t0 ) + y0;
1087                            z0 = x0 * x0;
1088                            t0 = z0 * ( qq1 + z0 * qq2 );
1089                            w0 = x0 * ( one + z0 * ( pp1 + z0 * pp2 ) );
1090                            j0 = ( ( ( j0 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
1091                            xsb0 = ( xsb0 >> 30 ) & 2;
1092                            n0 ^= ( xsb0 & ~( n0 << 1 ) );
1093                            xsb0 |= 1;
1094                            a0 = __vlibm_TBL_sincos_hi[j0+n0];
1095                            t0 = ( __vlibm_TBL_sincos_hi[j0+((n0+xsb0)&3)] * w0 + a0
1096                            *py0 = ( a0 + t0 );
1097                    }
1098            }

1100            if ( biguns )
1101                    __vlibm_vsin_big( nsave, xsave, sxsave, ysave, sysave, 0x413921f
1102 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   39377 Sun May 11 12:16:40 2014
new/usr/src/lib/libmvec/common/__vsincos.c
**********************************************************
```

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

#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  = 1.5707963267341256141660,  /* first 33 bits of pi/2 */
        pio2_2  = 6.0771005060396597659600e-11, /* second 33 bits of pi/2 */
        pio2_3  = 2.0222662487111664557960e-21, /* third 33 bits of pi/2 */
```



```
        pio2_3t = 8.4784276603688996439590e-32, /* pi/2 - pio2_3 */
        pp1             = -1.6666666666057604652762639431349825546760e-0001,
        pp2             =  8.3332612096909631267183765661461809444420e-0003,
        qq1             = -4.9999999999777109864070239559087115578700e-0001,
        qq2             =  4.1666548638572193506450558810188420895800e-0002,
        poly1[2]= { -1.6666666666662966980521513892030158965600e-0001,
                                -4.9999999999999317014640608788882945244810e-0001
        poly2[2]= {  8.3333333239095129568399345528033637666300e-0003,
                                 4.1666666663948619175356405939637082223190e-0002
        poly3[2]= { -1.9841262379979766927915517782300984039600e-0004,
                                -1.3888885526561428678327566873685168146200e-0003
        poly4[2]= {  2.7534036248542772376499876228483303511100e-0006,
                                 2.4785194236814607966181282894545305247590e-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;

        nsave = n;
        xsave = x;
        sxsave = stridex;
        ysave = y;
        sysave = stridey;
        csave = c;
        scsave = stridec;
        biguns = 0;

        do /* MAIN LOOP */
        {

                /* Gotos here so _break_ exits MAIN LOOP. */
LOOP0:  /* Find first arg in right range. */
                xsb0 = HI(x); /* get most significant word */
                hx0 = xsb0 & ~0x80000000; /* mask off sign bit */
                if ( hx0 > 0x3fe921fb ) {
                        /* Too big: arg reduction needed, so leave for second pa
                        biguns = 1;
                        x += stridex;
                        y += stridey;
                        c += stridec;
                        i = 0;
                        if ( --n <= 0 )
                                break;
```

```
128                         goto LOOP0;
129                 }
130                 if ( hx0 < 0x3e400000 ) {
131                         /* Too small.  cos x ~ 1, sin x ~ x. */
 61                         volatile int v = *x;
132                         *c = 1.0;
133                         *y = *x;
134                         x += stridex;
135                         y += stridey;
136                         c += stridec;
137                         i = 0;
138                         if ( --n <= 0 )
139                                 break;
140                         goto LOOP0;
141                 }
142                 x0 = *x;
143                 py0 = y;
144                 pc0 = c;
145                 x += stridex;
146                 y += stridey;
147                 c += stridec;
148                 i = 1;
149                 if ( --n <= 0 )
150                         break;

152 LOOP1: /* Get second arg, same as above. */
153                 xsb1 = HI(x);
154                 hx1 = xsb1 & ~0x80000000;
155                 if ( hx1 > 0x3fe921fb )
156                 {
157                         biguns = 1;
158                         x += stridex;
159                         y += stridey;
160                         c += stridec;
161                         i = 1;
162                         if ( --n <= 0 )
163                                 break;
164                         goto LOOP1;
165                 }
166                 if ( hx1 < 0x3e400000 )
167                 {
 98                         volatile int v = *x;
168                         *c = 1.0;
169                         *y = *x;
170                         x += stridex;
171                         y += stridey;
172                         c += stridec;
173                         i = 1;
174                         if ( --n <= 0 )
175                                 break;
176                         goto LOOP1;
177                 }
178                 x1 = *x;
179                 py1 = y;
180                 pc1 = c;
181                 x += stridex;
182                 y += stridey;
183                 c += stridec;
184                 i = 2;
185                 if ( --n <= 0 )
186                         break;

188 LOOP2: /* Get third arg, same as above. */
189                 xsb2 = HI(x);
190                 hx2 = xsb2 & ~0x80000000;
191                 if ( hx2 > 0x3fe921fb )
```



```
192                 {
193                         biguns = 1;
194                         x += stridex;
195                         y += stridey;
196                         c += stridec;
197                         i = 2;
198                         if ( --n <= 0 )
199                                 break;
200                         goto LOOP2;
201                 }
202                 if ( hx2 < 0x3e400000 )
203                 {
135                         volatile int v = *x;
204                         *c = 1.0;
205                         *y = *x;
206                         x += stridex;
207                         y += stridey;
208                         c += stridec;
209                         i = 2;
210                         if ( --n <= 0 )
211                                 break;
212                         goto LOOP2;
213                 }
214                 x2 = *x;
215                 py2 = y;
216                 pc2 = c;

218                 /*
219                  * 0x3fc40000 = 5/32 ~ 0.15625
220                  * Get msb after subtraction.  Will be 1 only if
221                  * hx0 - 5/32 is negative.
222                  */
223                 i = ( hx2 - 0x3fc40000 ) >> 31;
224                 i |= ( ( hx1 - 0x3fc40000 ) >> 30 ) & 2;
225                 i |= ( ( hx0 - 0x3fc40000 ) >> 29 ) & 4;
226                 switch ( i )
227                 {
228                         double          a1_0, a1_1, a1_2, a2_0, a2_1, a2_2;
229                         double          w0, w1, w2;
230                         double          t0, t1, t2, t1_0, t1_1, t1_2, t2_0, t2_1
231                         double          z0, z1, z2;
232                         unsigned        j0, j1, j2;

234                 case 0: /* All are > 5/32 */
235                         j0 = ( xsb0 + 0x4000 ) & 0xffff8000;
236                         j1 = ( xsb1 + 0x4000 ) & 0xffff8000;
237                         j2 = ( xsb2 + 0x4000 ) & 0xffff8000;

239                         HI(&t0) = j0;
240                         HI(&t1) = j1;
241                         HI(&t2) = j2;
242                         LO(&t0) = 0;
243                         LO(&t1) = 0;
244                         LO(&t2) = 0;

246                         x0 -= t0;
247                         x1 -= t1;
248                         x2 -= t2;

250                         z0 = x0 * x0;
251                         z1 = x1 * x1;
252                         z2 = x2 * x2;

254                         t0 = z0 * ( qq1 + z0 * qq2 );
255                         t1 = z1 * ( qq1 + z1 * qq2 );
256                         t2 = z2 * ( qq1 + z2 * qq2 );
```

```
258                         w0 = x0 * ( one + z0 * ( pp1 + z0 * pp2 ) );
259                         w1 = x1 * ( one + z1 * ( pp1 + z1 * pp2 ) );
260                         w2 = x2 * ( one + z2 * ( pp1 + z2 * pp2 ) );

262                         j0 = ( ( ( j0 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
263                         j1 = ( ( ( j1 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
264                         j2 = ( ( ( j2 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~

266                         xsb0 = ( xsb0 >> 30 ) & 2;
267                         xsb1 = ( xsb1 >> 30 ) & 2;
268                         xsb2 = ( xsb2 >> 30 ) & 2;

270                         a1_0 = __vlibm_TBL_sincos_hi[j0+xsb0]; /* sin_hi(t) */
271                         a1_1 = __vlibm_TBL_sincos_hi[j1+xsb1];
272                         a1_2 = __vlibm_TBL_sincos_hi[j2+xsb2];

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

282                         *pc0 = a2_0 + t2_0;
283                         *pc1 = a2_1 + t2_1;
284                         *pc2 = a2_2 + t2_2;

286                         t1_0 = a2_0*w0 + a1_0*t0;
287                         t1_1 = a2_1*w1 + a1_1*t1;
288                         t1_2 = a2_2*w2 + a1_2*t2;

290                         t1_0 += __vlibm_TBL_sincos_lo[j0+xsb0]; /* sin_lo(t) */
291                         t1_1 += __vlibm_TBL_sincos_lo[j1+xsb1];
292                         t1_2 += __vlibm_TBL_sincos_lo[j2+xsb2];

294                         *py0 = a1_0 + t1_0;
295                         *py1 = a1_1 + t1_1;
296                         *py2 = a1_2 + t1_2;

298                         break;

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



```
323                         a1_0 = __vlibm_TBL_sincos_hi[j0+xsb0]; /* sin_hi(t) */
324                         a1_1 = __vlibm_TBL_sincos_hi[j1+xsb1];

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

332                         *pc0 = a2_0 + t2_0;
333                         *pc1 = a2_1 + t2_1;
334                         *pc2 = one + t2;

336                         t1_0 = a2_0*w0 + a1_0*t0;
337                         t1_1 = a2_1*w1 + a1_1*t1;
338                         t2 = z2 * ( poly3[0] + z2 * poly4[0] );

340                         t1_0 += __vlibm_TBL_sincos_lo[j0+xsb0]; /* sin_lo(t) */
341                         t1_1 += __vlibm_TBL_sincos_lo[j1+xsb1];
342                         t2 = z2 * ( poly1[0] + z2 * ( poly2[0] + t2 ) );

344                         *py0 = a1_0 + t1_0;
345                         *py1 = a1_1 + t1_1;
346                         t2 = x2 + x2 * t2;
347                         *py2 = t2;

349                         break;

351                 case 2:
352                         j0 = ( xsb0 + 0x4000 ) & 0xffff8000;
353                         j2 = ( xsb2 + 0x4000 ) & 0xffff8000;
354                         HI(&t0) = j0;
355                         HI(&t2) = j2;
356                         LO(&t0) = 0;
357                         LO(&t2) = 0;
358                         x0 -= t0;
359                         x2 -= t2;
360                         z0 = x0 * x0;
361                         z1 = x1 * x1;
362                         z2 = x2 * x2;
363                         t0 = z0 * ( qq1 + z0 * qq2 );
364                         t1 = z1 * ( poly3[1] + z1 * poly4[1] );
365                         t2 = z2 * ( qq1 + z2 * qq2 );
366                         w0 = x0 * ( one + z0 * ( pp1 + z0 * pp2 ) );
367                         t1 = z1 * ( poly1[1] + z1 * ( poly2[1] + t1 ) );
368                         w2 = x2 * ( one + z2 * ( pp1 + z2 * pp2 ) );
369                         j0 = ( ( ( j0 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
370                         j2 = ( ( ( j2 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
371                         xsb0 = ( xsb0 >> 30 ) & 2;
372                         xsb2 = ( xsb2 >> 30 ) & 2;

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

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

383                         *pc0 = a2_0 + t2_0;
384                         *pc1 = one + t1;
385                         *pc2 = a2_2 + t2_2;

387                         t1_0 = a2_0*w0 + a1_0*t0;
388                         t1 = z1 * ( poly3[0] + z1 * poly4[0] );
```

```
389                         t1_2 = a2_2*w2 + a1_2*t2;

391                         t1_0 += __vlibm_TBL_sincos_lo[j0+xsb0]; /* sin_lo(t) */
392                         t1 = z1 * ( poly1[0] + z1 * ( poly2[0] + t1 ) );
393                         t1_2 += __vlibm_TBL_sincos_lo[j2+xsb2];

395                         *py0 = a1_0 + t1_0;
396                         t1 = x1 + x1 * t1;
397                         *py1 = t1;
398                         *py2 = a1_2 + t1_2;

400                         break;

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

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

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

424                         *pc0 = a2_0 + t2_0;
425                         *pc1 = one + t1;
426                         *pc2 = one + t2;

428                         t1_0 = a2_0*w0 + a1_0*t0;
429                         t1 = z1 * ( poly3[0] + z1 * poly4[0] );
430                         t2 = z2 * ( poly3[0] + z2 * poly4[0] );

432                         t1_0 += __vlibm_TBL_sincos_lo[j0+xsb0]; /* sin_lo(t) */
433                         t1 = z1 * ( poly1[0] + z1 * ( poly2[0] + t1 ) );
434                         t2 = z2 * ( poly1[0] + z2 * ( poly2[0] + t2 ) );

436                         *py0 = a1_0 + t1_0;
437                         t1 = x1 + x1 * t1;
438                         *py1 = t1;
439                         t2 = x2 + x2 * t2;
440                         *py2 = t2;

442                         break;

444                 case 4:
445                         j1 = ( xsb1 + 0x4000 ) & 0xffff8000;
446                         j2 = ( xsb2 + 0x4000 ) & 0xffff8000;
447                         HI(&t1) = j1;
448                         HI(&t2) = j2;
449                         LO(&t1) = 0;
450                         LO(&t2) = 0;
451                         x1 -= t1;
452                         x2 -= t2;
453                         z0 = x0 * x0;
454                         z1 = x1 * x1;
```



```
455                         z2 = x2 * x2;
456                         t0 = z0 * ( poly3[1] + z0 * poly4[1] );
457                         t1 = z1 * ( qq1 + z1 * qq2 );
458                         t2 = z2 * ( qq1 + z2 * qq2 );
459                         t0 = z0 * ( poly1[1] + z0 * ( poly2[1] + t0 ) );
460                         w1 = x1 * ( one + z1 * ( pp1 + z1 * pp2 ) );
461                         w2 = x2 * ( one + z2 * ( pp1 + z2 * pp2 ) );
462                         j1 = ( ( ( j1 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
463                         j2 = ( ( ( j2 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
464                         xsb1 = ( xsb1 >> 30 ) & 2;
465                         xsb2 = ( xsb2 >> 30 ) & 2;

467                         a1_1 = __vlibm_TBL_sincos_hi[j1+xsb1];
468                         a1_2 = __vlibm_TBL_sincos_hi[j2+xsb2];

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

476                         *pc0 = one + t0;
477                         *pc1 = a2_1 + t2_1;
478                         *pc2 = a2_2 + t2_2;

480                         t0 = z0 * ( poly3[0] + z0 * poly4[0] );
481                         t1_1 = a2_1*w1 + a1_1*t1;
482                         t1_2 = a2_2*w2 + a1_2*t2;

484                         t0 = z0 * ( poly1[0] + z0 * ( poly2[0] + t0 ) );
485                         t1_1 += __vlibm_TBL_sincos_lo[j1+xsb1];
486                         t1_2 += __vlibm_TBL_sincos_lo[j2+xsb2];

488                         t0 = x0 + x0 * t0;
489                         *py0 = t0;
490                         *py1 = a1_1 + t1_1;
491                         *py2 = a1_2 + t1_2;

493                         break;

495                 case 5:
496                         j1 = ( xsb1 + 0x4000 ) & 0xffff8000;
497                         HI(&t1) = j1;
498                         LO(&t1) = 0;
499                         x1 -= t1;
500                         z0 = x0 * x0;
501                         z1 = x1 * x1;
502                         z2 = x2 * x2;
503                         t0 = z0 * ( poly3[1] + z0 * poly4[1] );
504                         t1 = z1 * ( qq1 + z1 * qq2 );
505                         t2 = z2 * ( poly3[1] + z2 * poly4[1] );
506                         t0 = z0 * ( poly1[1] + z0 * ( poly2[1] + t0 ) );
507                         w1 = x1 * ( one + z1 * ( pp1 + z1 * pp2 ) );
508                         t2 = z2 * ( poly1[1] + z2 * ( poly2[1] + t2 ) );
509                         j1 = ( ( ( j1 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
510                         xsb1 = ( xsb1 >> 30 ) & 2;

512                         a1_1 = __vlibm_TBL_sincos_hi[j1+xsb1];

514                         a2_1 = __vlibm_TBL_sincos_hi[j1+1];

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

518                         *pc0 = one + t0;
519                         *pc1 = a2_1 + t2_1;
520                         *pc2 = one + t2;
```

```
522                         t0 = z0 * ( poly3[0] + z0 * poly4[0] );
523                         t1_1 = a2_1*w1 + a1_1*t1;
524                         t2 = z2 * ( poly3[0] + z2 * poly4[0] );

526                         t0 = z0 * ( poly1[0] + z0 * ( poly2[0] + t0 ) );
527                         t1_1 += __vlibm_TBL_sincos_lo[j1+xsb1];
528                         t2 = z2 * ( poly1[0] + z2 * ( poly2[0] + t2 ) );

530                         t0 = x0 + x0 * t0;
531                         *py0 = t0;
532                         *py1 = a1_1 + t1_1;
533                         t2 = x2 + x2 * t2;
534                         *py2 = t2;

536                         break;

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

556                         a2_2 = __vlibm_TBL_sincos_hi[j2+1];

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

560                         *pc0 = one + t0;
561                         *pc1 = one + t1;
562                         *pc2 = a2_2 + t2_2;

564                         t0 = z0 * ( poly3[0] + z0 * poly4[0] );
565                         t1 = z1 * ( poly3[0] + z1 * poly4[0] );
566                         t1_2 = a2_2*w2 + a1_2*t2;

568                         t0 = z0 * ( poly1[0] + z0 * ( poly2[0] + t0 ) );
569                         t1 = z1 * ( poly1[0] + z1 * ( poly2[0] + t1 ) );
570                         t1_2 += __vlibm_TBL_sincos_lo[j2+xsb2];

572                         t0 = x0 + x0 * t0;
573                         *py0 = t0;
574                         t1 = x1 + x1 * t1;
575                         *py1 = t1;
576                         *py2 = a1_2 + t1_2;

578                         break;

580                 case 7: /* All are < 5/32 */
581                         z0 = x0 * x0;
582                         z1 = x1 * x1;
583                         z2 = x2 * x2;
584                         t0 = z0 * ( poly3[1] + z0 * poly4[1] );
585                         t1 = z1 * ( poly3[1] + z1 * poly4[1] );
586                         t2 = z2 * ( poly3[1] + z2 * poly4[1] );
```



```
587                         t0 = z0 * ( poly1[1] + z0 * ( poly2[1] + t0 ) );
588                         t1 = z1 * ( poly1[1] + z1 * ( poly2[1] + t1 ) );
589                         t2 = z2 * ( poly1[1] + z2 * ( poly2[1] + t2 ) );
590                         *pc0 = one + t0;
591                         *pc1 = one + t1;
592                         *pc2 = one + t2;
593                         t0 = z0 * ( poly3[0] + z0 * poly4[0] );
594                         t1 = z1 * ( poly3[0] + z1 * poly4[0] );
595                         t2 = z2 * ( poly3[0] + z2 * poly4[0] );
596                         t0 = z0 * ( poly1[0] + z0 * ( poly2[0] + t0 ) );
597                         t1 = z1 * ( poly1[0] + z1 * ( poly2[0] + t1 ) );
598                         t2 = z2 * ( poly1[0] + z2 * ( poly2[0] + t2 ) );
599                         t0 = x0 + x0 * t0;
600                         t1 = x1 + x1 * t1;
601                         t2 = x2 + x2 * t2;
602                         *py0 = t0;
603                         *py1 = t1;
604                         *py2 = t2;
605                         break;
606                 }

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

614         /*
615          * CLEAN UP last 0, 1, or 2 elts.
616          */
617         if ( i > 0 ) /* Clean up elts at tail.  i < 3. */
618         {
619                 double          a1_0, a1_1, a2_0, a2_1;
620                 double          w0, w1;
621                 double          t0, t1, t1_0, t1_1, t2_0, t2_1;
622                 double          z0, z1;
623                 unsigned        j0, j1;

625                 if ( i > 1 )
626                 {
627                         if ( hx1 < 0x3fc40000 )
628                         {
629                                 z1 = x1 * x1;
630                                 t1 = z1 * ( poly3[1] + z1 * poly4[1] );
631                                 t1 = z1 * ( poly1[1] + z1 * ( poly2[1] + t1 ) );
632                                 t1 = one + t1;
633                                 *pc1 = t1;
634                                 t1 = z1 * ( poly3[0] + z1 * poly4[0] );
635                                 t1 = z1 * ( poly1[0] + z1 * ( poly2[0] + t1 ) );
636                                 t1 = x1 + x1 * t1;
637                                 *py1 = t1;
638                         }
639                         else
640                         {
641                                 j1 = ( xsb1 + 0x4000 ) & 0xffff8000;
642                                 HI(&t1) = j1;
643                                 LO(&t1) = 0;
644                                 x1 -= t1;
645                                 z1 = x1 * x1;
646                                 t1 = z1 * ( qq1 + z1 * qq2 );
647                                 w1 = x1 * ( one + z1 * ( pp1 + z1 * pp2 ) );
648                                 j1 = ( ( ( j1 & ~0x80000000 ) - 0x3fc40000 ) >>
649                                 xsb1 = ( xsb1 >> 30 ) & 2;
650                                 a1_1 = __vlibm_TBL_sincos_hi[j1+xsb1];
651                                 a2_1 = __vlibm_TBL_sincos_hi[j1+1];
652                                 t2_1 = __vlibm_TBL_sincos_lo[j1+1] - ( a1_1*w1 -
```

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

692         if ( !biguns )
693                 return;

695         /*
696          * Take care of BIGUNS.
697          */
698         n = nsave;
699         x = xsave;
700         stridex = sxsave;
701         y = ysave;
702         stridey = sysave;
703         c = csave;
704         stridec = scsave;
705         biguns = 0;

707         x0_or_one[1] = 1.0;
708         x1_or_one[1] = 1.0;
709         x2_or_one[1] = 1.0;
710         x0_or_one[3] = -1.0;
711         x1_or_one[3] = -1.0;
712         x2_or_one[3] = -1.0;
713         y0_or_zero[1] = 0.0;
714         y1_or_zero[1] = 0.0;
715         y2_or_zero[1] = 0.0;
716         y0_or_zero[3] = 0.0;
717         y1_or_zero[3] = 0.0;
718         y2_or_zero[3] = 0.0;
```



```
720         do
721         {
722                 double          fn0, fn1, fn2, a0, a1, a2, w0, w1, w2, y0, y1, y
723                 unsigned        hx;
724                 int                     n0, n1, n2;

726                 /*
727                  * Find 3 more to work on: Not already done, not too big.
728                  */
729 loop0:
730                 hx = HI(x);
731                 xsb0 = hx >> 31;
732                 hx &= ~0x80000000;
733                 if ( hx <= 0x3fe921fb ) /* Done above. */
734                 {
735                         x += stridex;
736                         y += stridey;
737                         c += stridec;
738                         i = 0;
739                         if ( --n <= 0 )
740                                 break;
741                         goto loop0;
742                 }
743                 if ( hx > 0x413921fb ) /* (1.6471e+06) Too big: leave it. */
744                 {
745                         if ( hx >= 0x7ff00000 ) /* Inf or NaN */
746                         {
747                                 x0 = *x;
748                                 *y = x0 - x0;
749                                 *c = x0 - x0;
750                         }
751                         else {
752                                 biguns = 1;
753                         }
754                         x += stridex;
755                         y += stridey;
756                         c += stridec;
757                         i = 0;
758                         if ( --n <= 0 )
759                                 break;
760                         goto loop0;
761                 }
762                 x0 = *x;
763                 py0 = y;
764                 pc0 = c;
765                 x += stridex;
766                 y += stridey;
767                 c += stridec;
768                 i = 1;
769                 if ( --n <= 0 )
770                         break;

772 loop1:
773                 hx = HI(x);
774                 xsb1 = hx >> 31;
775                 hx &= ~0x80000000;
776                 if ( hx <= 0x3fe921fb )
777                 {
778                         x += stridex;
779                         y += stridey;
780                         c += stridec;
781                         i = 1;
782                         if ( --n <= 0 )
783                                 break;
784                         goto loop1;
```

```
785                 }
786                 if ( hx > 0x413921fb )
787                 {
788                         if ( hx >= 0x7ff00000 )
789                         {
790                                 x1 = *x;
791                                 *y = x1 - x1;
792                                 *c = x1 - x1;
793                         }
794                         else {
795                                 biguns = 1;
796                         }
797                         x += stridex;
798                         y += stridey;
799                         c += stridec;
800                         i = 1;
801                         if ( --n <= 0 )
802                                 break;
803                         goto loop1;
804                 }
805                 x1 = *x;
806                 py1 = y;
807                 pc1 = c;
808                 x += stridex;
809                 y += stridey;
810                 c += stridec;
811                 i = 2;
812                 if ( --n <= 0 )
813                         break;

815 loop2:
816                 hx = HI(x);
817                 xsb2 = hx >> 31;
818                 hx &= ~0x80000000;
819                 if ( hx <= 0x3fe921fb )
820                 {
821                         x += stridex;
822                         y += stridey;
823                         c += stridec;
824                         i = 2;
825                         if ( --n <= 0 )
826                                 break;
827                         goto loop2;
828                 }
829                 if ( hx > 0x413921fb )
830                 {
831                         if ( hx >= 0x7ff00000 )
832                         {
833                                 x2 = *x;
834                                 *y = x2 - x2;
835                                 *c = x2 - x2;
836                         }
837                         else {
838                                 biguns = 1;
839                         }
840                         x += stridex;
841                         y += stridey;
842                         c += stridec;
843                         i = 2;
844                         if ( --n <= 0 )
845                                 break;
846                         goto loop2;
847                 }
848                 x2 = *x;
849                 py2 = y;
850                 pc2 = c;
```



```
852                 n0 = (int) ( x0 * invpio2 + half[xsb0] );
853                 n1 = (int) ( x1 * invpio2 + half[xsb1] );
854                 n2 = (int) ( x2 * invpio2 + half[xsb2] );
855                 fn0 = (double) n0;
856                 fn1 = (double) n1;
857                 fn2 = (double) n2;
858                 n0 &= 3;
859                 n1 &= 3;
860                 n2 &= 3;
861                 a0 = x0 - fn0 * pio2_1;
862                 a1 = x1 - fn1 * pio2_1;
863                 a2 = x2 - fn2 * pio2_1;
864                 w0 = fn0 * pio2_2;
865                 w1 = fn1 * pio2_2;
866                 w2 = fn2 * pio2_2;
867                 x0 = a0 - w0;
868                 x1 = a1 - w1;
869                 x2 = a2 - w2;
870                 y0 = ( a0 - x0 ) - w0;
871                 y1 = ( a1 - x1 ) - w1;
872                 y2 = ( a2 - x2 ) - w2;
873                 a0 = x0;
874                 a1 = x1;
875                 a2 = x2;
876                 w0 = fn0 * pio2_3 - y0;
877                 w1 = fn1 * pio2_3 - y1;
878                 w2 = fn2 * pio2_3 - y2;
879                 x0 = a0 - w0;
880                 x1 = a1 - w1;
881                 x2 = a2 - w2;
882                 y0 = ( a0 - x0 ) - w0;
883                 y1 = ( a1 - x1 ) - w1;
884                 y2 = ( a2 - x2 ) - w2;
885                 a0 = x0;
886                 a1 = x1;
887                 a2 = x2;
888                 w0 = fn0 * pio2_3t - y0;
889                 w1 = fn1 * pio2_3t - y1;
890                 w2 = fn2 * pio2_3t - y2;
891                 x0 = a0 - w0;
892                 x1 = a1 - w1;
893                 x2 = a2 - w2;
894                 y0 = ( a0 - x0 ) - w0;
895                 y1 = ( a1 - x1 ) - w1;
896                 y2 = ( a2 - x2 ) - w2;
897                 xsb2 = HI(&x2);
898                 i = ( ( xsb2 & ~0x80000000 ) - 0x3fc40000 ) >> 31;
899                 xsb1 = HI(&x1);
900                 i |= ( ( ( xsb1 & ~0x80000000 ) - 0x3fc40000 ) >> 30 ) & 2;
901                 xsb0 = HI(&x0);
902                 i |= ( ( ( xsb0 & ~0x80000000 ) - 0x3fc40000 ) >> 29 ) & 4;
903                 switch ( i )
904                 {
905                         double          a1_0, a1_1, a1_2, a2_0, a2_1, a2_2;
906                         double          t0, t1, t2, t1_0, t1_1, t1_2, t2_0, t2_1
907                         double          z0, z1, z2;
908                         unsigned        j0, j1, j2;

910                 case 0:
911                         j0 = ( xsb0 + 0x4000 ) & 0xffff8000;
912                         j1 = ( xsb1 + 0x4000 ) & 0xffff8000;
913                         j2 = ( xsb2 + 0x4000 ) & 0xffff8000;
914                         HI(&t0) = j0;
915                         HI(&t1) = j1;
916                         HI(&t2) = j2;
```

```
                        LO(&t0) = 0;
                        LO(&t1) = 0;
                        LO(&t2) = 0;
                        x0 = ( x0 - t0 ) + y0;
                        x1 = ( x1 - t1 ) + y1;
                        x2 = ( x2 - t2 ) + y2;
                        z0 = x0 * x0;
                        z1 = x1 * x1;
                        z2 = x2 * x2;
                        t0 = z0 * ( qq1 + z0 * qq2 );
                        t1 = z1 * ( qq1 + z1 * qq2 );
                        t2 = z2 * ( qq1 + z2 * qq2 );
                        w0 = x0 * ( one + z0 * ( pp1 + z0 * pp2 ) );
                        w1 = x1 * ( one + z1 * ( pp1 + z1 * pp2 ) );
                        w2 = x2 * ( one + z2 * ( pp1 + z2 * pp2 ) );
                        j0 = ( ( ( j0 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
                        j1 = ( ( ( j1 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
                        j2 = ( ( ( j2 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
                        xsb0 = ( xsb0 >> 30 ) & 2;
                        xsb1 = ( xsb1 >> 30 ) & 2;
                        xsb2 = ( xsb2 >> 30 ) & 2;
                        n0 ^= ( xsb0 & ~( n0 << 1 ) );
                        n1 ^= ( xsb1 & ~( n1 << 1 ) );
                        n2 ^= ( xsb2 & ~( n2 << 1 ) );
                        xsb0 |= 1;
                        xsb1 |= 1;
                        xsb2 |= 1;

                        a1_0 = __vlibm_TBL_sincos_hi[j0+n0];
                        a1_1 = __vlibm_TBL_sincos_hi[j1+n1];
                        a1_2 = __vlibm_TBL_sincos_hi[j2+n2];

                        a2_0 = __vlibm_TBL_sincos_hi[j0+((n0+xsb0)&3)];
                        a2_1 = __vlibm_TBL_sincos_hi[j1+((n1+xsb1)&3)];
                        a2_2 = __vlibm_TBL_sincos_hi[j2+((n2+xsb2)&3)];

                        t2_0 = __vlibm_TBL_sincos_lo[j0+((n0+xsb0)&3)] - ( a1_0*
                        t2_1 = __vlibm_TBL_sincos_lo[j1+((n1+xsb1)&3)] - ( a1_1*
                        t2_2 = __vlibm_TBL_sincos_lo[j2+((n2+xsb2)&3)] - ( a1_2*

                        w0 *= a2_0;
                        w1 *= a2_1;
                        w2 *= a2_2;

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

                        t1_0 = w0 + a1_0*t0;
                        t1_1 = w1 + a1_1*t1;
                        t1_2 = w2 + a1_2*t2;

                        t1_0 += __vlibm_TBL_sincos_lo[j0+n0];
                        t1_1 += __vlibm_TBL_sincos_lo[j1+n1];
                        t1_2 += __vlibm_TBL_sincos_lo[j2+n2];

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

                        break;

                case 1:
                        j0 = ( xsb0 + 0x4000 ) & 0xffff8000;
                        j1 = ( xsb1 + 0x4000 ) & 0xffff8000;
                        j2 = n2 & 1;
```



```
                        HI(&t0) = j0;
                        HI(&t1) = j1;
                        LO(&t0) = 0;
                        LO(&t1) = 0;
                        x2_or_one[0] = x2;
                        x2_or_one[2] = -x2;
                        x0 = ( x0 - t0 ) + y0;
                        x1 = ( x1 - t1 ) + y1;
                        y2_or_zero[0] = y2;
                        y2_or_zero[2] = -y2;
                        z0 = x0 * x0;
                        z1 = x1 * x1;
                        z2 = x2 * x2;
                        t0 = z0 * ( qq1 + z0 * qq2 );
                        t1 = z1 * ( qq1 + z1 * qq2 );
                        t2 = z2 * ( poly3[j2] + z2 * poly4[j2] );
                        w0 = x0 * ( one + z0 * ( pp1 + z0 * pp2 ) );
                        w1 = x1 * ( one + z1 * ( pp1 + z1 * pp2 ) );
                        t2 = z2 * ( poly1[j2] + z2 * ( poly2[j2] + t2 ) );
                        j0 = ( ( ( j0 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
                        j1 = ( ( ( j1 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
                        xsb0 = ( xsb0 >> 30 ) & 2;
                        xsb1 = ( xsb1 >> 30 ) & 2;
                        n0 ^= ( xsb0 & ~( n0 << 1 ) );
                        n1 ^= ( xsb1 & ~( n1 << 1 ) );
                        xsb0 |= 1;
                        xsb1 |= 1;
                        a1_0 = __vlibm_TBL_sincos_hi[j0+n0];
                        a1_1 = __vlibm_TBL_sincos_hi[j1+n1];

                        a2_0 = __vlibm_TBL_sincos_hi[j0+((n0+xsb0)&3)];
                        a2_1 = __vlibm_TBL_sincos_hi[j1+((n1+xsb1)&3)];

                        t2_0 = __vlibm_TBL_sincos_lo[j0+((n0+xsb0)&3)] - ( a1_0*
                        t2_1 = __vlibm_TBL_sincos_lo[j1+((n1+xsb1)&3)] - ( a1_1*
                        t2 = x2_or_one[n2] + ( y2_or_zero[n2] + x2_or_one[n2] *

                        *pc0 = a2_0 + t2_0;
                        *pc1 = a2_1 + t2_1;
                        *py2 = t2;

                        n2 = (n2 + 1) & 3;
                        j2 = (j2 + 1) & 1;
                        t2 = z2 * ( poly3[j2] + z2 * poly4[j2] );

                        t1_0 = a2_0*w0 + a1_0*t0;
                        t1_1 = a2_1*w1 + a1_1*t1;
                        t2 = z2 * ( poly1[j2] + z2 * ( poly2[j2] + t2 ) );

                        t1_0 += __vlibm_TBL_sincos_lo[j0+n0];
                        t1_1 += __vlibm_TBL_sincos_lo[j1+n1];
                        t2 = x2_or_one[n2] + ( y2_or_zero[n2] + x2_or_one[n2] *

                        *py0 = a1_0 + t1_0;
                        *py1 = a1_1 + t1_1;
                        *pc2 = t2;

                        break;

                case 2:
                        j0 = ( xsb0 + 0x4000 ) & 0xffff8000;
                        j1 = n1 & 1;
                        j2 = ( xsb2 + 0x4000 ) & 0xffff8000;
                        HI(&t0) = j0;
                        HI(&t2) = j2;
                        LO(&t0) = 0;
```

```
1049                        LO(&t2) = 0;
1050                        x1_or_one[0] = x1;
1051                        x1_or_one[2] = -x1;
1052                        x0 = ( x0 - t0 ) + y0;
1053                        y1_or_zero[0] = y1;
1054                        y1_or_zero[2] = -y1;
1055                        x2 = ( x2 - t2 ) + y2;
1056                        z0 = x0 * x0;
1057                        z1 = x1 * x1;
1058                        z2 = x2 * x2;
1059                        t0 = z0 * ( qq1 + z0 * qq2 );
1060                        t1 = z1 * ( poly3[j1] + z1 * poly4[j1] );
1061                        t2 = z2 * ( qq1 + z2 * qq2 );
1062                        w0 = x0 * ( one + z0 * ( pp1 + z0 * pp2 ) );
1063                        t1 = z1 * ( poly1[j1] + z1 * ( poly2[j1] + t1 ) );
1064                        w2 = x2 * ( one + z2 * ( pp1 + z2 * pp2 ) );
1065                        j0 = ( ( ( j0 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
1066                        j2 = ( ( ( j2 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
1067                        xsb0 = ( xsb0 >> 30 ) & 2;
1068                        xsb2 = ( xsb2 >> 30 ) & 2;
1069                        n0 ^= ( xsb0 & ~( n0 << 1 ) );
1070                        n2 ^= ( xsb2 & ~( n2 << 1 ) );
1071                        xsb0 |= 1;
1072                        xsb2 |= 1;

1074                        a1_0 = __vlibm_TBL_sincos_hi[j0+n0];
1075                        a1_2 = __vlibm_TBL_sincos_hi[j2+n2];

1077                        a2_0 = __vlibm_TBL_sincos_hi[j0+((n0+xsb0)&3)];
1078                        a2_2 = __vlibm_TBL_sincos_hi[j2+((n2+xsb2)&3)];

1080                        t2_0 = __vlibm_TBL_sincos_lo[j0+((n0+xsb0)&3)] - ( a1_0*
1081                        t1 = x1_or_one[n1] + ( y1_or_zero[n1] + x1_or_one[n1] *
1082                        t2_2 = __vlibm_TBL_sincos_lo[j2+((n2+xsb2)&3)] - ( a1_2*

1084                        *pc0 = a2_0 + t2_0;
1085                        *py1 = t1;
1086                        *pc2 = a2_2 + t2_2;

1088                        n1 = (n1 + 1) & 3;
1089                        j1 = (j1 + 1) & 1;
1090                        t1 = z1 * ( poly3[j1] + z1 * poly4[j1] );

1092                        t1_0 = a2_0*w0 + a1_0*t0;
1093                        t1 = z1 * ( poly1[j1] + z1 * ( poly2[j1] + t1 ) );
1094                        t1_2 = a2_2*w2 + a1_2*t2;

1096                        t1_0 += __vlibm_TBL_sincos_lo[j0+n0];
1097                        t1 = x1_or_one[n1] + ( y1_or_zero[n1] + x1_or_one[n1] *
1098                        t1_2 += __vlibm_TBL_sincos_lo[j2+n2];

1100                        *py0 = a1_0 + t1_0;
1101                        *pc1 = t1;
1102                        *py2 = a1_2 + t1_2;

1104                        break;

1106                case 3:
1107                        j0 = ( xsb0 + 0x4000 ) & 0xffff8000;
1108                        j1 = n1 & 1;
1109                        j2 = n2 & 1;
1110                        HI(&t0) = j0;
1111                        LO(&t0) = 0;
1112                        x1_or_one[0] = x1;
1113                        x1_or_one[2] = -x1;
1114                        x2_or_one[0] = x2;
```



```
1115                        x2_or_one[2] = -x2;
1116                        x0 = ( x0 - t0 ) + y0;
1117                        y1_or_zero[0] = y1;
1118                        y1_or_zero[2] = -y1;
1119                        y2_or_zero[0] = y2;
1120                        y2_or_zero[2] = -y2;
1121                        z0 = x0 * x0;
1122                        z1 = x1 * x1;
1123                        z2 = x2 * x2;
1124                        t0 = z0 * ( qq1 + z0 * qq2 );
1125                        t1 = z1 * ( poly3[j1] + z1 * poly4[j1] );
1126                        t2 = z2 * ( poly3[j2] + z2 * poly4[j2] );
1127                        w0 = x0 * ( one + z0 * ( pp1 + z0 * pp2 ) );
1128                        t1 = z1 * ( poly1[j1] + z1 * ( poly2[j1] + t1 ) );
1129                        t2 = z2 * ( poly1[j2] + z2 * ( poly2[j2] + t2 ) );
1130                        j0 = ( ( ( j0 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
1131                        xsb0 = ( xsb0 >> 30 ) & 2;
1132                        n0 ^= ( xsb0 & ~( n0 << 1 ) );
1133                        xsb0 |= 1;

1135                        a1_0 = __vlibm_TBL_sincos_hi[j0+n0];
1136                        a2_0 = __vlibm_TBL_sincos_hi[j0+((n0+xsb0)&3)];

1138                        t2_0 = __vlibm_TBL_sincos_lo[j0+((n0+xsb0)&3)] - ( a1_0*
1139                        t1 = x1_or_one[n1] + ( y1_or_zero[n1] + x1_or_one[n1] *
1140                        t2 = x2_or_one[n2] + ( y2_or_zero[n2] + x2_or_one[n2] *

1142                        *pc0 = a2_0 + t2_0;
1143                        *py1 = t1;
1144                        *py2 = t2;

1146                        n1 = (n1 + 1) & 3;
1147                        n2 = (n2 + 1) & 3;
1148                        j1 = (j1 + 1) & 1;
1149                        j2 = (j2 + 1) & 1;

1151                        t1_0 = a2_0*w0 + a1_0*t0;
1152                        t1 = z1 * ( poly3[j1] + z1 * poly4[j1] );
1153                        t2 = z2 * ( poly3[j2] + z2 * poly4[j2] );

1155                        t1_0 += __vlibm_TBL_sincos_lo[j0+n0];
1156                        t1 = z1 * ( poly1[j1] + z1 * ( poly2[j1] + t1 ) );
1157                        t2 = z2 * ( poly1[j2] + z2 * ( poly2[j2] + t2 ) );

1159                        t1 = x1_or_one[n1] + ( y1_or_zero[n1] + x1_or_one[n1] *
1160                        t2 = x2_or_one[n2] + ( y2_or_zero[n2] + x2_or_one[n2] *

1162                        *py0 = a1_0 + t1_0;
1163                        *pc1 = t1;
1164                        *pc2 = t2;

1166                        break;

1168                case 4:
1169                        j0 = n0 & 1;
1170                        j1 = ( xsb1 + 0x4000 ) & 0xffff8000;
1171                        j2 = ( xsb2 + 0x4000 ) & 0xffff8000;
1172                        HI(&t1) = j1;
1173                        HI(&t2) = j2;
1174                        LO(&t1) = 0;
1175                        LO(&t2) = 0;
1176                        x0_or_one[0] = x0;
1177                        x0_or_one[2] = -x0;
1178                        y0_or_zero[0] = y0;
1179                        y0_or_zero[2] = -y0;
1180                        x1 = ( x1 - t1 ) + y1;
```

```
1181                        x2 = ( x2 - t2 ) + y2;
1182                        z0 = x0 * x0;
1183                        z1 = x1 * x1;
1184                        z2 = x2 * x2;
1185                        t0 = z0 * ( poly3[j0] + z0 * poly4[j0] );
1186                        t1 = z1 * ( qq1 + z1 * qq2 );
1187                        t2 = z2 * ( qq1 + z2 * qq2 );
1188                        t0 = z0 * ( poly1[j0] + z0 * ( poly2[j0] + t0 ) );
1189                        w1 = x1 * ( one + z1 * ( pp1 + z1 * pp2 ) );
1190                        w2 = x2 * ( one + z2 * ( pp1 + z2 * pp2 ) );
1191                        j1 = ( ( ( j1 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
1192                        j2 = ( ( ( j2 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
1193                        xsb1 = ( xsb1 >> 30 ) & 2;
1194                        xsb2 = ( xsb2 >> 30 ) & 2;
1195                        n1 ^= ( xsb1 & ~( n1 << 1 ) );
1196                        n2 ^= ( xsb2 & ~( n2 << 1 ) );
1197                        xsb1 |= 1;
1198                        xsb2 |= 1;

1200                        a1_1 = __vlibm_TBL_sincos_hi[j1+n1];
1201                        a1_2 = __vlibm_TBL_sincos_hi[j2+n2];

1203                        a2_1 = __vlibm_TBL_sincos_hi[j1+((n1+xsb1)&3)];
1204                        a2_2 = __vlibm_TBL_sincos_hi[j2+((n2+xsb2)&3)];

1206                        t0 = x0_or_one[n0] + ( y0_or_zero[n0] + x0_or_one[n0] *
1207                        t2_1 = __vlibm_TBL_sincos_lo[j1+((n1+xsb1)&3)] - ( a1_1*
1208                        t2_2 = __vlibm_TBL_sincos_lo[j2+((n2+xsb2)&3)] - ( a1_2*

1210                        *py0 = t0;
1211                        *pc1 = a2_1 + t2_1;
1212                        *pc2 = a2_2 + t2_2;

1214                        n0 = (n0 + 1) & 3;
1215                        j0 = (j0 + 1) & 1;
1216                        t0 = z0 * ( poly3[j0] + z0 * poly4[j0] );

1218                        t0 = z0 * ( poly1[j0] + z0 * ( poly2[j0] + t0 ) );
1219                        t1_1 = a2_1*w1 + a1_1*t1;
1220                        t1_2 = a2_2*w2 + a1_2*t2;

1222                        t0 = x0_or_one[n0] + ( y0_or_zero[n0] + x0_or_one[n0] *
1223                        t1_1 += __vlibm_TBL_sincos_lo[j1+n1];
1224                        t1_2 += __vlibm_TBL_sincos_lo[j2+n2];

1226                        *py1 = a1_1 + t1_1;
1227                        *py2 = a1_2 + t1_2;
1228                        *pc0 = t0;

1230                        break;

1232                case 5:
1233                        j0 = n0 & 1;
1234                        j1 = ( xsb1 + 0x4000 ) & 0xffff8000;
1235                        j2 = n2 & 1;
1236                        HI(&t1) = j1;
1237                        LO(&t1) = 0;
1238                        x0_or_one[0] = x0;
1239                        x0_or_one[2] = -x0;
1240                        x2_or_one[0] = x2;
1241                        x2_or_one[2] = -x2;
1242                        y0_or_zero[0] = y0;
1243                        y0_or_zero[2] = -y0;
1244                        x1 = ( x1 - t1 ) + y1;
1245                        y2_or_zero[0] = y2;
1246                        y2_or_zero[2] = -y2;
```



```
1247                        z0 = x0 * x0;
1248                        z1 = x1 * x1;
1249                        z2 = x2 * x2;
1250                        t0 = z0 * ( poly3[j0] + z0 * poly4[j0] );
1251                        t1 = z1 * ( qq1 + z1 * qq2 );
1252                        t2 = z2 * ( poly3[j2] + z2 * poly4[j2] );
1253                        t0 = z0 * ( poly1[j0] + z0 * ( poly2[j0] + t0 ) );
1254                        w1 = x1 * ( one + z1 * ( pp1 + z1 * pp2 ) );
1255                        t2 = z2 * ( poly1[j2] + z2 * ( poly2[j2] + t2 ) );
1256                        j1 = ( ( ( j1 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
1257                        xsb1 = ( xsb1 >> 30 ) & 2;
1258                        n1 ^= ( xsb1 & ~( n1 << 1 ) );
1259                        xsb1 |= 1;

1261                        a1_1 = __vlibm_TBL_sincos_hi[j1+n1];
1262                        a2_1 = __vlibm_TBL_sincos_hi[j1+((n1+xsb1)&3)];

1264                        t0 = x0_or_one[n0] + ( y0_or_zero[n0] + x0_or_one[n0] *
1265                        t2_1 = __vlibm_TBL_sincos_lo[j1+((n1+xsb1)&3)] - ( a1_1*
1266                        t2 = x2_or_one[n2] + ( y2_or_zero[n2] + x2_or_one[n2] *

1268                        *py0 = t0;
1269                        *pc1 = a2_1 + t2_1;
1270                        *py2 = t2;

1272                        n0 = (n0 + 1) & 3;
1273                        n2 = (n2 + 1) & 3;
1274                        j0 = (j0 + 1) & 1;
1275                        j2 = (j2 + 1) & 1;

1277                        t0 = z0 * ( poly3[j0] + z0 * poly4[j0] );
1278                        t1_1 = a2_1*w1 + a1_1*t1;
1279                        t2 = z2 * ( poly3[j2] + z2 * poly4[j2] );

1281                        t0 = z0 * ( poly1[j0] + z0 * ( poly2[j0] + t0 ) );
1282                        t1_1 += __vlibm_TBL_sincos_lo[j1+n1];
1283                        t2 = z2 * ( poly1[j2] + z2 * ( poly2[j2] + t2 ) );

1285                        t0 = x0_or_one[n0] + ( y0_or_zero[n0] + x0_or_one[n0] *
1286                        t2 = x2_or_one[n2] + ( y2_or_zero[n2] + x2_or_one[n2] *

1288                        *pc0 = t0;
1289                        *py1 = a1_1 + t1_1;
1290                        *pc2 = t2;

1292                        break;

1294                case 6:
1295                        j0 = n0 & 1;
1296                        j1 = n1 & 1;
1297                        j2 = ( xsb2 + 0x4000 ) & 0xffff8000;
1298                        HI(&t2) = j2;
1299                        LO(&t2) = 0;
1300                        x0_or_one[0] = x0;
1301                        x0_or_one[2] = -x0;
1302                        x1_or_one[0] = x1;
1303                        x1_or_one[2] = -x1;
1304                        y0_or_zero[0] = y0;
1305                        y0_or_zero[2] = -y0;
1306                        y1_or_zero[0] = y1;
1307                        y1_or_zero[2] = -y1;
1308                        x2 = ( x2 - t2 ) + y2;
1309                        z0 = x0 * x0;
1310                        z1 = x1 * x1;
1311                        z2 = x2 * x2;
1312                        t0 = z0 * ( poly3[j0] + z0 * poly4[j0] );
```

```
1313                        t1 = z1 * ( poly3[j1] + z1 * poly4[j1] );
1314                        t2 = z2 * ( qq1 + z2 * qq2 );
1315                        t0 = z0 * ( poly1[j0] + z0 * ( poly2[j0] + t0 ) );
1316                        t1 = z1 * ( poly1[j1] + z1 * ( poly2[j1] + t1 ) );
1317                        w2 = x2 * ( one + z2 * ( pp1 + z2 * pp2 ) );
1318                        j2 = ( ( ( j2 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
1319                        xsb2 = ( xsb2 >> 30 ) & 2;
1320                        n2 ^= ( xsb2 & ~( n2 << 1 ) );
1321                        xsb2 |= 1;

1323                        a1_2 = __vlibm_TBL_sincos_hi[j2+n2];
1324                        a2_2 = __vlibm_TBL_sincos_hi[j2+((n2+xsb2)&3)];

1326                        t0 = x0_or_one[n0] + ( y0_or_zero[n0] + x0_or_one[n0] *
1327                        t1 = x1_or_one[n1] + ( y1_or_zero[n1] + x1_or_one[n1] *
1328                        t2_2 = __vlibm_TBL_sincos_lo[j2+((n2+xsb2)&3)] - ( a1_2*

1330                        *py0 = t0;
1331                        *py1 = t1;
1332                        *pc2 = a2_2 + t2_2;

1334                        n0 = (n0 + 1) & 3;
1335                        n1 = (n1 + 1) & 3;
1336                        j0 = (j0 + 1) & 1;
1337                        j1 = (j1 + 1) & 1;

1339                        t0 = z0 * ( poly3[j0] + z0 * poly4[j0] );
1340                        t1 = z1 * ( poly3[j1] + z1 * poly4[j1] );
1341                        t1_2 = a2_2*w2 + a1_2*t2;

1343                        t0 = z0 * ( poly1[j0] + z0 * ( poly2[j0] + t0 ) );
1344                        t1 = z1 * ( poly1[j1] + z1 * ( poly2[j1] + t1 ) );
1345                        t1_2 += __vlibm_TBL_sincos_lo[j2+n2];

1347                        t0 = x0_or_one[n0] + ( y0_or_zero[n0] + x0_or_one[n0] *
1348                        t1 = x1_or_one[n1] + ( y1_or_zero[n1] + x1_or_one[n1] *

1350                        *pc0 = t0;
1351                        *pc1 = t1;
1352                        *py2 = a1_2 + t1_2;

1354                        break;

1356                case 7:
1357                        j0 = n0 & 1;
1358                        j1 = n1 & 1;
1359                        j2 = n2 & 1;
1360                        x0_or_one[0] = x0;
1361                        x0_or_one[2] = -x0;
1362                        x1_or_one[0] = x1;
1363                        x1_or_one[2] = -x1;
1364                        x2_or_one[0] = x2;
1365                        x2_or_one[2] = -x2;
1366                        y0_or_zero[0] = y0;
1367                        y0_or_zero[2] = -y0;
1368                        y1_or_zero[0] = y1;
1369                        y1_or_zero[2] = -y1;
1370                        y2_or_zero[0] = y2;
1371                        y2_or_zero[2] = -y2;
1372                        z0 = x0 * x0;
1373                        z1 = x1 * x1;
1374                        z2 = x2 * x2;
1375                        t0 = z0 * ( poly3[j0] + z0 * poly4[j0] );
1376                        t1 = z1 * ( poly3[j1] + z1 * poly4[j1] );
1377                        t2 = z2 * ( poly3[j2] + z2 * poly4[j2] );
1378                        t0 = z0 * ( poly1[j0] + z0 * ( poly2[j0] + t0 ) );
```



```
1379                        t1 = z1 * ( poly1[j1] + z1 * ( poly2[j1] + t1 ) );
1380                        t2 = z2 * ( poly1[j2] + z2 * ( poly2[j2] + t2 ) );
1381                        t0 = x0_or_one[n0] + ( y0_or_zero[n0] + x0_or_one[n0] *
1382                        t1 = x1_or_one[n1] + ( y1_or_zero[n1] + x1_or_one[n1] *
1383                        t2 = x2_or_one[n2] + ( y2_or_zero[n2] + x2_or_one[n2] *
1384                        *py0 = t0;
1385                        *py1 = t1;
1386                        *py2 = t2;

1388                        n0 = (n0 + 1) & 3;
1389                        n1 = (n1 + 1) & 3;
1390                        n2 = (n2 + 1) & 3;
1391                        j0 = (j0 + 1) & 1;
1392                        j1 = (j1 + 1) & 1;
1393                        j2 = (j2 + 1) & 1;
1394                        t0 = z0 * ( poly3[j0] + z0 * poly4[j0] );
1395                        t1 = z1 * ( poly3[j1] + z1 * poly4[j1] );
1396                        t2 = z2 * ( poly3[j2] + z2 * poly4[j2] );
1397                        t0 = z0 * ( poly1[j0] + z0 * ( poly2[j0] + t0 ) );
1398                        t1 = z1 * ( poly1[j1] + z1 * ( poly2[j1] + t1 ) );
1399                        t2 = z2 * ( poly1[j2] + z2 * ( poly2[j2] + t2 ) );
1400                        t0 = x0_or_one[n0] + ( y0_or_zero[n0] + x0_or_one[n0] *
1401                        t1 = x1_or_one[n1] + ( y1_or_zero[n1] + x1_or_one[n1] *
1402                        t2 = x2_or_one[n2] + ( y2_or_zero[n2] + x2_or_one[n2] *
1403                        *pc0 = t0;
1404                        *pc1 = t1;
1405                        *pc2 = t2;
1406                        break;
1407                }

1409                x += stridex;
1410                y += stridey;
1411                c += stridec;
1412                i = 0;
1413        } while ( --n > 0 );

1415        if ( i > 0 )
1416        {
1417                double          a1_0, a1_1, a2_0, a2_1;
1418                double          t0, t1, t1_0, t1_1, t2_0, t2_1;
1419                double          fn0, fn1, a0, a1, w0, w1, y0, y1;
1420                double          z0, z1;
1421                unsigned        j0, j1;
1422                int             n0, n1;

1424                if ( i > 1 )
1425                {
1426                        n1 = (int) ( x1 * invpio2 + half[xsb1] );
1427                        fn1 = (double) n1;
1428                        n1 &= 3;
1429                        a1 = x1 - fn1 * pio2_1;
1430                        w1 = fn1 * pio2_2;
1431                        x1 = a1 - w1;
1432                        y1 = ( a1 - x1 ) - w1;
1433                        a1 = x1;
1434                        w1 = fn1 * pio2_3 - y1;
1435                        x1 = a1 - w1;
1436                        y1 = ( a1 - x1 ) - w1;
1437                        a1 = x1;
1438                        w1 = fn1 * pio2_3t - y1;
1439                        x1 = a1 - w1;
1440                        y1 = ( a1 - x1 ) - w1;
1441                        xsb1 = HI(&x1);
1442                        if ( ( xsb1 & ~0x80000000 ) < 0x3fc40000 )
1443                        {
1444                                j1 = n1 & 1;
```

```
1445                                    x1_or_one[0] = x1;
1446                                    x1_or_one[2] = -x1;
1447                                    y1_or_zero[0] = y1;
1448                                    y1_or_zero[2] = -y1;
1449                                    z1 = x1 * x1;
1450                                    t1 = z1 * ( poly3[j1] + z1 * poly4[j1] );
1451                                    t1 = z1 * ( poly1[j1] + z1 * ( poly2[j1] + t1 )
1452                                    t1 = x1_or_one[n1] + ( y1_or_zero[n1] + x1_or_on
1453                                    *py1 = t1;
1454                                    n1 = (n1 + 1) & 3;
1455                                    j1 = (j1 + 1) & 1;
1456                                    t1 = z1 * ( poly3[j1] + z1 * poly4[j1] );
1457                                    t1 = z1 * ( poly1[j1] + z1 * ( poly2[j1] + t1 )
1458                                    t1 = x1_or_one[n1] + ( y1_or_zero[n1] + x1_or_on
1459                                    *pc1 = t1;
1460                            }
1461                            else
1462                            {
1463                                    j1 = ( xsb1 + 0x4000 ) & 0xffff8000;
1464                                    HI(&t1) = j1;
1465                                    LO(&t1) = 0;
1466                                    x1 = ( x1 - t1 ) + y1;
1467                                    z1 = x1 * x1;
1468                                    t1 = z1 * ( qq1 + z1 * qq2 );
1469                                    w1 = x1 * ( one + z1 * ( pp1 + z1 * pp2 ) );
1470                                    j1 = ( ( ( j1 & ~0x80000000 ) - 0x3fc40000 ) >>
1471                                    xsb1 = ( xsb1 >> 30 ) & 2;
1472                                    n1 ^= ( xsb1 & ~( n1 << 1 ) );
1473                                    xsb1 |= 1;
1474                                    a1_1 = __vlibm_TBL_sincos_hi[j1+n1];
1475                                    a2_1 = __vlibm_TBL_sincos_hi[j1+((n1+xsb1)&3)];
1476                                    t2_1 = __vlibm_TBL_sincos_lo[j1+((n1+xsb1)&3)] -
1477                                    *pc1 = a2_1 + t2_1;
1478                                    t1_1 = a2_1*w1 + a1_1*t1;
1479                                    t1_1 += __vlibm_TBL_sincos_lo[j1+n1];
1480                                    *py1 = a1_1 + t1_1;
1481                            }
1482                    }
1483                    n0 = (int) ( x0 * invpio2 + half[xsb0] );
1484                    fn0 = (double) n0;
1485                    n0 &= 3;
1486                    a0 = x0 - fn0 * pio2_1;
1487                    w0 = fn0 * pio2_2;
1488                    x0 = a0 - w0;
1489                    y0 = ( a0 - x0 ) - w0;
1490                    a0 = x0;
1491                    w0 = fn0 * pio2_3 - y0;
1492                    x0 = a0 - w0;
1493                    y0 = ( a0 - x0 ) - w0;
1494                    a0 = x0;
1495                    w0 = fn0 * pio2_3t - y0;
1496                    x0 = a0 - w0;
1497                    y0 = ( a0 - x0 ) - w0;
1498                    xsb0 = HI(&x0);
1499                    if ( ( xsb0 & ~0x80000000 ) < 0x3fc40000 )
1500                    {
1501                            j0 = n0 & 1;
1502                            x0_or_one[0] = x0;
1503                            x0_or_one[2] = -x0;
1504                            y0_or_zero[0] = y0;
1505                            y0_or_zero[2] = -y0;
1506                            z0 = x0 * x0;
1507                            t0 = z0 * ( poly3[j0] + z0 * poly4[j0] );
1508                            t0 = z0 * ( poly1[j0] + z0 * ( poly2[j0] + t0 ) );
1509                            t0 = x0_or_one[n0] + ( y0_or_zero[n0] + x0_or_one[n0] *
1510                            *py0 = t0;
```



```
1511                            n0 = (n0 + 1) & 3;
1512                            j0 = (j0 + 1) & 1;
1513                            t0 = z0 * ( poly3[j0] + z0 * poly4[j0] );
1514                            t0 = z0 * ( poly1[j0] + z0 * ( poly2[j0] + t0 ) );
1515                            t0 = x0_or_one[n0] + ( y0_or_zero[n0] + x0_or_one[n0] *
1516                            *pc0 = t0;
1517                    }
1518                    else
1519                    {
1520                            j0 = ( xsb0 + 0x4000 ) & 0xffff8000;
1521                            HI(&t0) = j0;
1522                            LO(&t0) = 0;
1523                            x0 = ( x0 - t0 ) + y0;
1524                            z0 = x0 * x0;
1525                            t0 = z0 * ( qq1 + z0 * qq2 );
1526                            w0 = x0 * ( one + z0 * ( pp1 + z0 * pp2 ) );
1527                            j0 = ( ( ( j0 & ~0x80000000 ) - 0x3fc40000 ) >> 13 ) & ~
1528                            xsb0 = ( xsb0 >> 30 ) & 2;
1529                            n0 ^= ( xsb0 & ~( n0 << 1 ) );
1530                            xsb0 |= 1;
1531                            a1_0 = __vlibm_TBL_sincos_hi[j0+n0];
1532                            a2_0 = __vlibm_TBL_sincos_hi[j0+((n0+xsb0)&3)];
1533                            t2_0 = __vlibm_TBL_sincos_lo[j0+((n0+xsb0)&3)] - ( a1_0*
1534                            *pc0 = a2_0 + t2_0;
1535                            t1_0 = a2_0*w0 + a1_0*t0;
1536                            t1_0 += __vlibm_TBL_sincos_lo[j0+n0];
1537                            *py0 = a1_0 + t1_0;
1538                    }
1539            }

1541            if ( biguns ) {
1542                    __vlibm_vsincos_big( nsave, xsave, sxsave, ysave, sysave, csave,
1543            }
1544 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
    7042 Sun May 11 12:16:42 2014
new/usr/src/lib/libmvec/common/__vsincosf.c
**********************************************************
______unchanged_portion_omitted_

  76 #define S0      C[0]
  77 #define S1      C[1]
  78 #define S2      C[2]
  79 #define one     C[3]
  80 #define mhalf   C[4]
  81 #define C0      C[5]
  82 #define C1      C[6]
  83 #define C2      C[7]
  84 #define invpio2 C[8]
  85 #define c3two51 C[9]
  86 #define pio2_1  C[10]
  87 #define pio2_t  C[11]

  89 #define PREPROCESS(N, sindex, cindex, label)                          \
  90         hx = *(int *)x;                                              \
  91         ix = hx & 0x7fffffff;                                        \
  92         t = *x;                                                      \
  93         x += stridex;                                                \
  94         if (ix <= 0x3f490fdb) { /* |x| < pi/4 */                     \
  95                 if (ix == 0) {                                       \
  96                         s[sindex] = t;                               \
  97                         c[cindex] = one;                             \
  98                         goto label;                                  \
  99                 }                                                    \
 100                 y##N = (double)t;                                    \
 101                 n##N = 0;                                            \
 102         } else if (ix <= 0x49c90fdb) { /* |x| < 2^19*pi */           \
 103                 y##N = (double)t;                                    \
 104                 medium = 1;                                          \
 105         } else {                                                     \
 106                 if (ix >= 0x7f800000) { /* inf or nan */             \
 107                         s[sindex] = c[cindex] = t / t;               \
 108                         goto label;                                  \
 109                 }                                                    \
 110                 z##N = y##N = (double)t;                             \
 111                 hx = HI(y##N);                                       \
 112                 n##N = ((hx >> 20) & 0x7ff) - 1046;                  \
 113                 HI(z##N) = (hx & 0xfffff) | 0x41600000;              \
 114                 n##N = __vlibm_rem_pio2m(&z##N, &y##N, n##N, 1, 0);  \
 115                 if (hx < 0) {                                        \
 116                         y##N = -y##N;                                \
 117                         n##N = -n##N;                                \
 118                 }                                                    \
 119                 z##N = y##N * y##N;                                  \
 120                 f##N = (float)(y##N + y##N * z##N * (S0 + z##N *     \
 121                     (S1 + z##N * S2)));                              \
 122                 g##N = (float)(one + z##N * (mhalf + z##N * (C0 +    \
 123                    z##N * (C1 + z##N * C2))));                       \
 124                 if (n##N & 2) {                                      \
 125                         f##N = -f##N;                                \
 126                         g##N = -g##N;                                \
 127                 }                                                    \
 128                 if (n##N & 1) {                                      \
 129                         s[sindex] = g##N;                            \
 130                         c[cindex] = -f##N;                           \
 131                 } else {                                             \
 132                         s[sindex] = f##N;                            \
 133                         c[cindex] = g##N;                            \
 134                 }                                                    \
 135                 goto label;                                          \
```



```
 136         }

 138 #define PROCESS(N)                                                    \
 139         if (medium) {                                                \
 140                 z##N = y##N * invpio2 + c3two51;                     \
 141                 n##N = LO(z##N);                                     \
 142                 z##N -= c3two51;                                     \
 143                 y##N = (y##N - z##N * pio2_1) - z##N * pio2_t;       \
 144         }                                                            \
 145         z##N = y##N * y##N;                                          \
 146         f##N = (float)(y##N + y##N * z##N * (S0 + z##N * (S1 + z##N * S2)));\
 147         g##N = (float)(one + z##N * (mhalf + z##N * (C0 + z##N *      \
 148             (C1 + z##N * C2))));                                     \
 149         if (n##N & 2) {                                              \
 150                 f##N = -f##N;                                        \
 151                 g##N = -g##N;                                        \
 152         }                                                            \
 153         if (n##N & 1) {                                              \
 154                 *s = g##N;                                           \
 155                 *c = -f##N;                                          \
 156         } else {                                                     \
 157                 *s = f##N;                                           \
 158                 *c = g##N;                                           \
 159         }                                                            \
 160         s += strides;                                                \
 161         c += stridec

 163 void
 164 __vsincosf(int n, float *restrict x, int stridex,
 165     float *restrict s, int strides, float *restrict c, int stridec)
 166 {
 167         double          y0, y1, y2, y3;
 168         double          z0, z1, z2, z3;
 169         float           f0, f1, f2, f3, t;
 170         float           g0, g1, g2, g3;
 171         int             n0 = 0, n1 = 0, n2 = 0, n3, hx, ix, medium;
 171         int             n0, n1, n2, n3, hx, ix, medium;

 173         s -= strides;
 174         c -= stridec;

 176         for (;;) {
 177 begin:
 178                 s += strides;
 179                 c += stridec;

 181                 if (--n < 0)
 182                         break;

 184                 medium = 0;
 185                 PREPROCESS(0, 0, 0, begin);

 187                 if (--n < 0)
 188                         goto process1;

 190                 PREPROCESS(1, strides, stridec, process1);

 192                 if (--n < 0)
 193                         goto process2;

 195                 PREPROCESS(2, (strides << 1), (stridec << 1), process2);

 197                 if (--n < 0)
 198                         goto process3;

 200                 PREPROCESS(3, (strides << 1) + strides,
```

```
201                     (stridec << 1) + stridec, process3);

203                 if (medium) {
204                         z0 = y0 * invpio2 + c3two51;
205                         z1 = y1 * invpio2 + c3two51;
206                         z2 = y2 * invpio2 + c3two51;
207                         z3 = y3 * invpio2 + c3two51;

209                         n0 = LO(z0);
210                         n1 = LO(z1);
211                         n2 = LO(z2);
212                         n3 = LO(z3);

214                         z0 -= c3two51;
215                         z1 -= c3two51;
216                         z2 -= c3two51;
217                         z3 -= c3two51;

219                         y0 = (y0 - z0 * pio2_1) - z0 * pio2_t;
220                         y1 = (y1 - z1 * pio2_1) - z1 * pio2_t;
221                         y2 = (y2 - z2 * pio2_1) - z2 * pio2_t;
222                         y3 = (y3 - z3 * pio2_1) - z3 * pio2_t;
223                 }

225                 z0 = y0 * y0;
226                 z1 = y1 * y1;
227                 z2 = y2 * y2;
228                 z3 = y3 * y3;

230                 f0 = (float)(y0 + y0 * z0 * (S0 + z0 * (S1 + z0 * S2)));
231                 f1 = (float)(y1 + y1 * z1 * (S0 + z1 * (S1 + z1 * S2)));
232                 f2 = (float)(y2 + y2 * z2 * (S0 + z2 * (S1 + z2 * S2)));
233                 f3 = (float)(y3 + y3 * z3 * (S0 + z3 * (S1 + z3 * S2)));

235                 g0 = (float)(one + z0 * (mhalf + z0 * (C0 + z0 *
236                     (C1 + z0 * C2))));
237                 g1 = (float)(one + z1 * (mhalf + z1 * (C0 + z1 *
238                     (C1 + z1 * C2))));
239                 g2 = (float)(one + z2 * (mhalf + z2 * (C0 + z2 *
240                     (C1 + z2 * C2))));
241                 g3 = (float)(one + z3 * (mhalf + z3 * (C0 + z3 *
242                     (C1 + z3 * C2))));

244                 if (n0 & 2) {
245                         f0 = -f0;
246                         g0 = -g0;
247                 }
248                 if (n1 & 2) {
249                         f1 = -f1;
250                         g1 = -g1;
251                 }
252                 if (n2 & 2) {
253                         f2 = -f2;
254                         g2 = -g2;
255                 }
256                 if (n3 & 2) {
257                         f3 = -f3;
258                         g3 = -g3;
259                 }

261                 if (n0 & 1) {
262                         *s = g0;
263                         *c = -f0;
264                 } else {
265                         *s = f0;
266                         *c = g0;
```



```
267                 }
268                 s += strides;
269                 c += stridec;

271                 if (n1 & 1) {
272                         *s = g1;
273                         *c = -f1;
274                 } else {
275                         *s = f1;
276                         *c = g1;
277                 }
278                 s += strides;
279                 c += stridec;

281                 if (n2 & 1) {
282                         *s = g2;
283                         *c = -f2;
284                 } else {
285                         *s = f2;
286                         *c = g2;
287                 }
288                 s += strides;
289                 c += stridec;

291                 if (n3 & 1) {
292                         *s = g3;
293                         *c = -f3;
294                 } else {
295                         *s = f3;
296                         *c = g3;
297                 }
298                 continue;

300 process1:
301                 PROCESS(0);
302                 continue;

304 process2:
305                 PROCESS(0);
306                 PROCESS(1);
307                 continue;

309 process3:
310                 PROCESS(0);
311                 PROCESS(1);
312                 PROCESS(2);
313         }
314 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   10486 Sun May 11 12:16:43 2014
new/usr/src/lib/libmvec/common/__vsinf.c
**********************************************************
______unchanged_portion_omitted_

  76 #define S0      C[0]
  77 #define S1      C[1]
  78 #define S2      C[2]
  79 #define one     C[3]
  80 #define mhalf   C[4]
  81 #define C0      C[5]
  82 #define C1      C[6]
  83 #define C2      C[7]
  84 #define invpio2 C[8]
  85 #define c3two51 C[9]
  86 #define pio2_1  C[10]
  87 #define pio2_t  C[11]

  89 #define PREPROCESS(N, index, label)                                   \
  90         hx = *(int *)x;                                              \
  91         ix = hx & 0x7fffffff;                                        \
  92         t = *x;                                                      \
  93         x += stridex;                                                \
  94         if (ix <= 0x3f490fdb) { /* |x| < pi/4 */                     \
  95                 if (ix == 0) {                                       \
  96                         y[index] = t;                                \
  97                         goto label;                                  \
  98                 }                                                    \
  99                 y##N = (double)t;                                    \
 100                 n##N = 0;                                            \
 101         } else if (ix <= 0x49c90fdb) { /* |x| < 2^19*pi */           \
 102                 y##N = (double)t;                                    \
 103                 medium = 1;                                          \
 104         } else {                                                     \
 105                 if (ix >= 0x7f800000) { /* inf or nan */             \
 106                         y[index] = t / t;                            \
 107                         goto label;                                  \
 108                 }                                                    \
 109                 z##N = y##N = (double)t;                             \
 110                 hx = HI(y##N);                                       \
 111                 n##N = ((hx >> 20) & 0x7ff) - 1046;                  \
 112                 HI(z##N) = (hx & 0xfffff) | 0x41600000;              \
 113                 n##N = __vlibm_rem_pio2m(&z##N, &y##N, n##N, 1, 0);  \
 114                 if (hx < 0) {                                        \
 115                         y##N = -y##N;                                \
 116                         n##N = -n##N;                                \
 117                 }                                                    \
 118                 z##N = y##N * y##N;                                  \
 119                 if (n##N & 1) { /* compute cos y */                  \
 120                         f##N = (float)(one + z##N * (mhalf + z##N *  \
 121                             (C0 + z##N * (C1 + z##N * C2))));        \
 122                 } else { /* compute sin y */                         \
 123                         f##N = (float)(y##N + y##N * z##N * (S0 +    \
 124                             z##N * (S1 + z##N * S2)));               \
 125                 }                                                    \
 126                 y[index] = (n##N & 2)? -f##N : f##N;                 \
 127                 goto label;                                          \
 128         }

 130 #define PROCESS(N)                                                    \
 131         if (medium) {                                                \
 132                 z##N = y##N * invpio2 + c3two51;                     \
 133                 n##N = LO(z##N);                                     \
 134                 z##N -= c3two51;                                     \
 135                 y##N = (y##N - z##N * pio2_1) - z##N * pio2_t;       \
```



```
 136         }                                                            \
 137         z##N = y##N * y##N;                                          \
 138         if (n##N & 1) { /* compute cos y */                           \
 139                 f##N = (float)(one + z##N * (mhalf + z##N * (C0 +    \
 140                     z##N * (C1 + z##N * C2))));                      \
 141         } else { /* compute sin y */                                 \
 142                 f##N = (float)(y##N + y##N * z##N * (S0 + z##N * (S1 +  \
 143                     z##N * S2)));                                    \
 144         }                                                            \
 145         *y = (n##N & 2)? -f##N : f##N;                               \
 146         y += stridey

 148 void
 149 __vsinf(int n, float *restrict x, int stridex, float *restrict y,
 150     int stridey)
 151 {
 152         double          y0, y1, y2, y3;
 153         double          z0, z1, z2, z3;
 154         float           f0, f1, f2, f3, t;
 155         int             n0 = 0, n1 = 0, n2 = 0, n3, hx, ix, medium;
 155         int             n0, n1, n2, n3, hx, ix, medium;

 157         y -= stridey;

 159         for (;;) {
 160 begin:
 161                 y += stridey;

 163                 if (--n < 0)
 164                         break;

 166                 medium = 0;
 167                 PREPROCESS(0, 0, begin);

 169                 if (--n < 0)
 170                         goto process1;

 172                 PREPROCESS(1, stridey, process1);

 174                 if (--n < 0)
 175                         goto process2;

 177                 PREPROCESS(2, (stridey << 1), process2);

 179                 if (--n < 0)
 180                         goto process3;

 182                 PREPROCESS(3, (stridey << 1) + stridey, process3);

 184                 if (medium) {
 185                         z0 = y0 * invpio2 + c3two51;
 186                         z1 = y1 * invpio2 + c3two51;
 187                         z2 = y2 * invpio2 + c3two51;
 188                         z3 = y3 * invpio2 + c3two51;

 190                         n0 = LO(z0);
 191                         n1 = LO(z1);
 192                         n2 = LO(z2);
 193                         n3 = LO(z3);

 195                         z0 -= c3two51;
 196                         z1 -= c3two51;
 197                         z2 -= c3two51;
 198                         z3 -= c3two51;

 200                         y0 = (y0 - z0 * pio2_1) - z0 * pio2_t;
```

```
201                         y1 = (y1 - z1 * pio2_1) - z1 * pio2_t;
202                         y2 = (y2 - z2 * pio2_1) - z2 * pio2_t;
203                         y3 = (y3 - z3 * pio2_1) - z3 * pio2_t;
204                 }
205
206                 z0 = y0 * y0;
207                 z1 = y1 * y1;
208                 z2 = y2 * y2;
209                 z3 = y3 * y3;
210
211                 hx = (n0 & 1) | ((n1 & 1) << 1) | ((n2 & 1) << 2) |
212                     ((n3 & 1) << 3);
213                 switch (hx) {
214                 case 0:
215                         f0 = (float)(y0 + y0 * z0 * (S0 + z0 * (S1 + z0 * S2)));
216                         f1 = (float)(y1 + y1 * z1 * (S0 + z1 * (S1 + z1 * S2)));
217                         f2 = (float)(y2 + y2 * z2 * (S0 + z2 * (S1 + z2 * S2)));
218                         f3 = (float)(y3 + y3 * z3 * (S0 + z3 * (S1 + z3 * S2)));
219                         break;
220
221                 case 1:
222                         f0 = (float)(one + z0 * (mhalf + z0 * (C0 +
223                             z0 * (C1 + z0 * C2))));
224                         f1 = (float)(y1 + y1 * z1 * (S0 + z1 * (S1 + z1 * S2)));
225                         f2 = (float)(y2 + y2 * z2 * (S0 + z2 * (S1 + z2 * S2)));
226                         f3 = (float)(y3 + y3 * z3 * (S0 + z3 * (S1 + z3 * S2)));
227                         break;
228
229                 case 2:
230                         f0 = (float)(y0 + y0 * z0 * (S0 + z0 * (S1 + z0 * S2)));
231                         f1 = (float)(one + z1 * (mhalf + z1 * (C0 +
232                             z1 * (C1 + z1 * C2))));
233                         f2 = (float)(y2 + y2 * z2 * (S0 + z2 * (S1 + z2 * S2)));
234                         f3 = (float)(y3 + y3 * z3 * (S0 + z3 * (S1 + z3 * S2)));
235                         break;
236
237                 case 3:
238                         f0 = (float)(one + z0 * (mhalf + z0 * (C0 +
239                             z0 * (C1 + z0 * C2))));
240                         f1 = (float)(one + z1 * (mhalf + z1 * (C0 +
241                             z1 * (C1 + z1 * C2))));
242                         f2 = (float)(y2 + y2 * z2 * (S0 + z2 * (S1 + z2 * S2)));
243                         f3 = (float)(y3 + y3 * z3 * (S0 + z3 * (S1 + z3 * S2)));
244                         break;
245
246                 case 4:
247                         f0 = (float)(y0 + y0 * z0 * (S0 + z0 * (S1 + z0 * S2)));
248                         f1 = (float)(y1 + y1 * z1 * (S0 + z1 * (S1 + z1 * S2)));
249                         f2 = (float)(one + z2 * (mhalf + z2 * (C0 +
250                             z2 * (C1 + z2 * C2))));
251                         f3 = (float)(y3 + y3 * z3 * (S0 + z3 * (S1 + z3 * S2)));
252                         break;
253
254                 case 5:
255                         f0 = (float)(one + z0 * (mhalf + z0 * (C0 +
256                             z0 * (C1 + z0 * C2))));
257                         f1 = (float)(y1 + y1 * z1 * (S0 + z1 * (S1 + z1 * S2)));
258                         f2 = (float)(one + z2 * (mhalf + z2 * (C0 +
259                             z2 * (C1 + z2 * C2))));
260                         f3 = (float)(y3 + y3 * z3 * (S0 + z3 * (S1 + z3 * S2)));
261                         break;
262
263                 case 6:
264                         f0 = (float)(y0 + y0 * z0 * (S0 + z0 * (S1 + z0 * S2)));
265                         f1 = (float)(one + z1 * (mhalf + z1 * (C0 +
266                             z1 * (C1 + z1 * C2))));
```



```
267                         f2 = (float)(one + z2 * (mhalf + z2 * (C0 +
268                             z2 * (C1 + z2 * C2))));
269                         f3 = (float)(y3 + y3 * z3 * (S0 + z3 * (S1 + z3 * S2)));
270                         break;
271
272                 case 7:
273                         f0 = (float)(one + z0 * (mhalf + z0 * (C0 +
274                             z0 * (C1 + z0 * C2))));
275                         f1 = (float)(one + z1 * (mhalf + z1 * (C0 +
276                             z1 * (C1 + z1 * C2))));
277                         f2 = (float)(one + z2 * (mhalf + z2 * (C0 +
278                             z2 * (C1 + z2 * C2))));
279                         f3 = (float)(y3 + y3 * z3 * (S0 + z3 * (S1 + z3 * S2)));
280                         break;
281
282                 case 8:
283                         f0 = (float)(y0 + y0 * z0 * (S0 + z0 * (S1 + z0 * S2)));
284                         f1 = (float)(y1 + y1 * z1 * (S0 + z1 * (S1 + z1 * S2)));
285                         f2 = (float)(y2 + y2 * z2 * (S0 + z2 * (S1 + z2 * S2)));
286                         f3 = (float)(one + z3 * (mhalf + z3 * (C0 +
287                             z3 * (C1 + z3 * C2))));
288                         break;
289
290                 case 9:
291                         f0 = (float)(one + z0 * (mhalf + z0 * (C0 +
292                             z0 * (C1 + z0 * C2))));
293                         f1 = (float)(y1 + y1 * z1 * (S0 + z1 * (S1 + z1 * S2)));
294                         f2 = (float)(y2 + y2 * z2 * (S0 + z2 * (S1 + z2 * S2)));
295                         f3 = (float)(one + z3 * (mhalf + z3 * (C0 +
296                             z3 * (C1 + z3 * C2))));
297                         break;
298
299                 case 10:
300                         f0 = (float)(y0 + y0 * z0 * (S0 + z0 * (S1 + z0 * S2)));
301                         f1 = (float)(one + z1 * (mhalf + z1 * (C0 +
302                             z1 * (C1 + z1 * C2))));
303                         f2 = (float)(y2 + y2 * z2 * (S0 + z2 * (S1 + z2 * S2)));
304                         f3 = (float)(one + z3 * (mhalf + z3 * (C0 +
305                             z3 * (C1 + z3 * C2))));
306                         break;
307
308                 case 11:
309                         f0 = (float)(one + z0 * (mhalf + z0 * (C0 +
310                             z0 * (C1 + z0 * C2))));
311                         f1 = (float)(one + z1 * (mhalf + z1 * (C0 +
312                             z1 * (C1 + z1 * C2))));
313                         f2 = (float)(y2 + y2 * z2 * (S0 + z2 * (S1 + z2 * S2)));
314                         f3 = (float)(one + z3 * (mhalf + z3 * (C0 +
315                             z3 * (C1 + z3 * C2))));
316                         break;
317
318                 case 12:
319                         f0 = (float)(y0 + y0 * z0 * (S0 + z0 * (S1 + z0 * S2)));
320                         f1 = (float)(y1 + y1 * z1 * (S0 + z1 * (S1 + z1 * S2)));
321                         f2 = (float)(one + z2 * (mhalf + z2 * (C0 +
322                             z2 * (C1 + z2 * C2))));
323                         f3 = (float)(one + z3 * (mhalf + z3 * (C0 +
324                             z3 * (C1 + z3 * C2))));
325                         break;
326
327                 case 13:
328                         f0 = (float)(one + z0 * (mhalf + z0 * (C0 +
329                             z0 * (C1 + z0 * C2))));
330                         f1 = (float)(y1 + y1 * z1 * (S0 + z1 * (S1 + z1 * S2)));
331                         f2 = (float)(one + z2 * (mhalf + z2 * (C0 +
332                             z2 * (C1 + z2 * C2))));
```

```
333                         f3 = (float)(one + z3 * (mhalf + z3 * (C0 +
334                             z3 * (C1 + z3 * C2))));
335                         break;

337                 case 14:
338                         f0 = (float)(y0 + y0 * z0 * (S0 + z0 * (S1 + z0 * S2)));
339                         f1 = (float)(one + z1 * (mhalf + z1 * (C0 +
340                             z1 * (C1 + z1 * C2))));
341                         f2 = (float)(one + z2 * (mhalf + z2 * (C0 +
342                             z2 * (C1 + z2 * C2))));
343                         f3 = (float)(one + z3 * (mhalf + z3 * (C0 +
344                             z3 * (C1 + z3 * C2))));
345                         break;

347                 default:
348                         f0 = (float)(one + z0 * (mhalf + z0 * (C0 +
349                             z0 * (C1 + z0 * C2))));
350                         f1 = (float)(one + z1 * (mhalf + z1 * (C0 +
351                             z1 * (C1 + z1 * C2))));
352                         f2 = (float)(one + z2 * (mhalf + z2 * (C0 +
353                             z2 * (C1 + z2 * C2))));
354                         f3 = (float)(one + z3 * (mhalf + z3 * (C0 +
355                             z3 * (C1 + z3 * C2))));
356                 }

358                 *y = (n0 & 2)? -f0 : f0;
359                 y += stridey;
360                 *y = (n1 & 2)? -f1 : f1;
361                 y += stridey;
362                 *y = (n2 & 2)? -f2 : f2;
363                 y += stridey;
364                 *y = (n3 & 2)? -f3 : f3;
365                 continue;

367 process1:
368                 PROCESS(0);
369                 continue;

371 process2:
372                 PROCESS(0);
373                 PROCESS(1);
374                 continue;

376 process3:
377                 PROCESS(0);
378                 PROCESS(1);
379                 PROCESS(2);
380         }
381 }
```

_____***unchanged_portion_omitted_***