```
**********************************************************
   71419 Sun May  4 03:04:40 2014
new/usr/src/Targetdirs
**********************************************************
```

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

  21 #
  22 # Copyright (c) 1989, 2010, Oracle and/or its affiliates. All rights reserved.
  23 # Copyright 2011, Richard Lowe
  24 # Copyright (c) 2012 by Delphix. All rights reserved.
  25 # Copyright (c) 2012, Igor Kozhukhov <ikozhukhov@gmail.com>
  26 #endif /* ! codereview */
  27 # Copyright 2012 OmniTI Computer Consulting, Inc.  All rights reserved.
  28 # Copyright (c) 2013 RackTop Systems.
  29 # Copyright 2013 Nexenta Systems, Inc.  All rights reserved.
  30 #

  32 #
  33 # It is easier to think in terms of directory names without the ROOT macro
  34 # prefix.  ROOTDIRS is TARGETDIRS with ROOT prefixes.  It is necessary
  35 # to work with ROOT prefixes when controlling conditional assignments.
  36 #

  38 DIRLINKS=       $(SYM.DIRS)
  39 $(BUILD64)      DIRLINKS += $(SYM.DIRS64)

  41 FILELINKS= $(SYM.USRCCSLIB) $(SYM.USRLIB)
  42 $(BUILD64)      FILELINKS += $(SYM.USRCCSLIB64) $(SYM.USRLIB64)

  44 TARGETDIRS=     $(DIRS)
  45 $(BUILD64)      TARGETDIRS += $(DIRS64)

  47 TARGETDIRS      += $(FILELINKS) $(DIRLINKS)

  49 i386_DIRS=                       \
  50         /boot/acpi               \
  51         /boot/acpi/tables        \
  52         /boot/grub               \
  53         /boot/grub/bin           \
  54         /platform/i86pc          \
  55         /lib/libmvec             \
  56         /usr/lib/xen             \
  57         /usr/lib/xen/bin

  59 sparc_DIRS=                              \
  60         /usr/lib/ldoms

  62 sparc_64ONLY= $(POUND_SIGN)
```



```
  63 64ONLY=  $($(MACH)_64ONLY)

  65 $(64ONLY) MACH32_DIRS=/usr/ucb/$(MACH32)

  67 DIRS= \
  68         /boot \
  69         /boot/solaris \
  70         /boot/solaris/bin \
  71         $($(MACH)_DIRS) \
  72         /dev \
  73         /dev/dsk \
  74         /dev/fd \
  75         /dev/ipnet \
  76         /dev/net \
  77         /dev/rdsk \
  78         /dev/rmt \
  79         /dev/pts \
  80         /dev/sad \
  81         /dev/swap \
  82         /dev/term \
  83         /dev/vt \
  84         /dev/zcons \
  85         /devices \
  86         /devices/pseudo \
  87         /etc \
  88         /etc/brand  \
  89         /etc/brand/solaris10  \
  90         /etc/cron.d \
  91         /etc/crypto \
  92         /etc/crypto/certs \
  93         /etc/crypto/crls \
  94         /etc/dbus-1 \
  95         /etc/dbus-1/system.d \
  96         /etc/default \
  97         /etc/devices  \
  98         /etc/dev  \
  99         /etc/dfs  \
 100         /etc/dladm \
 101         /etc/fs  \
 102         /etc/fs/nfs  \
 103         /etc/fs/zfs \
 104         /etc/ftpd  \
 105         /etc/hal \
 106         /etc/hal/fdi \
 107         /etc/hal/fdi/information \
 108         /etc/hal/fdi/information/10freedesktop \
 109         /etc/hal/fdi/information/20thirdparty \
 110         /etc/hal/fdi/information/30user \
 111         /etc/hal/fdi/policy \
 112         /etc/hal/fdi/policy/10osvendor \
 113         /etc/hal/fdi/policy/20thirdparty \
 114         /etc/hal/fdi/policy/30user \
 115         /etc/hal/fdi/preprobe \
 116         /etc/hal/fdi/preprobe/10osvendor \
 117         /etc/hal/fdi/preprobe/20thirdparty \
 118         /etc/hal/fdi/preprobe/30user \
 119         /etc/ipadm \
 120         /etc/iscsi \
 121         /etc/rpcsec     \
 122         /etc/security   \
 123         /etc/security/auth_attr.d \
 124         /etc/security/exec_attr.d \
 125         /etc/security/prof_attr.d \
 126         /etc/security/tsol      \
 127         /etc/gss        \
 128         /etc/init.d  \
```

```
129         /etc/dhcp       \
130         /etc/lib  \
131         /etc/mail  \
132         /etc/mail/cf  \
133         /etc/mail/cf/cf  \
134         /etc/mail/cf/domain  \
135         /etc/mail/cf/feature  \
136         /etc/mail/cf/m4  \
137         /etc/mail/cf/mailer  \
138         /etc/mail/cf/ostype  \
139         /etc/mail/cf/sh  \
140         /etc/net-snmp \
141         /etc/net-snmp/snmp \
142         /etc/opt  \
143         /etc/rc0.d  \
144         /etc/rc1.d  \
145         /etc/rc2.d  \
146         /etc/rc3.d  \
147         /etc/rcS.d  \
148         /etc/saf \
149         /etc/sasl       \
150         /etc/sfw \
151         /etc/skel       \
152         /etc/svc \
153         /etc/svc/profile \
154         /etc/svc/profile/site \
155         /etc/svc/volatile \
156         /etc/tm  \
157         /etc/usb   \
158         /etc/user_attr.d \
159         /etc/zfs  \
160         /etc/zones  \
161         /export  \
162         /home  \
163         /lib \
164         /lib/crypto \
165         /lib/inet \
166         /lib/fm \
167         /lib/secure \
168         /lib/svc \
169         /lib/svc/bin \
170         /lib/svc/capture \
171         /lib/svc/manifest \
172         /lib/svc/manifest/milestone \
173         /lib/svc/manifest/device \
174         /lib/svc/manifest/system \
175         /lib/svc/manifest/system/device \
176         /lib/svc/manifest/system/filesystem \
177         /lib/svc/manifest/system/security \
178         /lib/svc/manifest/system/svc \
179         /lib/svc/manifest/network \
180         /lib/svc/manifest/network/dns \
181         /lib/svc/manifest/network/ipsec \
182         /lib/svc/manifest/network/ldap \
183         /lib/svc/manifest/network/nfs \
184         /lib/svc/manifest/network/nis \
185         /lib/svc/manifest/network/rpc \
186         /lib/svc/manifest/network/security \
187         /lib/svc/manifest/network/shares \
188         /lib/svc/manifest/network/ssl \
189         /lib/svc/manifest/application \
190         /lib/svc/manifest/application/management \
191         /lib/svc/manifest/application/security \
192         /lib/svc/manifest/application/print \
193         /lib/svc/manifest/platform \
194         /lib/svc/manifest/platform/sun4u \
```



```
195         /lib/svc/manifest/platform/sun4v \
196         /lib/svc/manifest/site \
197         /lib/svc/method \
198         /lib/svc/monitor \
199         /lib/svc/seed \
200         /lib/svc/share \
201         /kernel  \
202         /mnt  \
203         /opt  \
204         /platform  \
205         /proc  \
206         /root  \
207         /sbin  \
208         /system \
209         /system/contract \
210         /system/object \
211         /tmp \
212         /usr  \
213         /usr/4lib \
214         /usr/ast \
215         /usr/ast/bin \
216         /usr/bin \
217         /usr/bin/$(MACH32) \
218         /usr/ccs \
219         /usr/ccs/bin \
220         /usr/ccs/lib \
221         /usr/demo \
222         /usr/demo/SOUND \
223         /usr/games \
224         /usr/has \
225         /usr/has/bin \
226         /usr/has/lib \
227         /usr/has/man \
228         /usr/has/man/man1has \
229         /usr/include \
230         /usr/include/ast \
231         /usr/include/fm \
232         /usr/include/gssapi \
233         /usr/include/hal \
234         /usr/include/kerberosv5 \
235         /usr/include/libmilter \
236         /usr/include/libpolkit \
237         /usr/include/sasl \
238         /usr/include/scsi \
239         /usr/include/security \
240         /usr/include/sys/crypto \
241         /usr/include/tsol \
242         /usr/kernel  \
243         /usr/kvm \
244         /usr/lib \
245         /usr/lib/abi \
246         /usr/lib/brand \
247         /usr/lib/brand/ipkg \
248         /usr/lib/brand/labeled \
249         /usr/lib/brand/shared \
250         /usr/lib/brand/sn1 \
251         /usr/lib/brand/solaris10 \
252         /usr/lib/class \
253         /usr/lib/class/FSS \
254         /usr/lib/class/FX \
255         /usr/lib/class/IA \
256         /usr/lib/class/RT \
257         /usr/lib/class/SDC \
258         /usr/lib/class/TS \
259         /usr/lib/crypto \
260         /usr/lib/drv \
```

```
261         /usr/lib/elfedit \
262         /usr/lib/fm \
263         /usr/lib/font \
264         /usr/lib/fs \
265         /usr/lib/fs/nfs \
266         /usr/lib/fs/proc \
267         /usr/lib/fs/smb \
268         /usr/lib/fs/zfs \
269         /usr/lib/gss \
270         /usr/lib/hal \
271         /usr/lib/inet \
272         /usr/lib/inet/dhcp \
273         /usr/lib/inet/dhcp/nsu \
274         /usr/lib/inet/dhcp/svc \
275         /usr/lib/inet/dhcp/svcadm \
276         /usr/lib/inet/ilb \
277         /usr/lib/inet/$(MACH32) \
278         /usr/lib/inet/wanboot \
279         /usr/lib/krb5 \
280         /usr/lib/link_audit \
281         /usr/lib/libp \
282         /usr/lib/lwp \
283         /usr/lib/mdb \
284         /usr/lib/mdb/kvm \
285         /usr/lib/mdb/proc \
286         /usr/lib/nfs \
287         /usr/net \
288         /usr/net/servers \
289         /usr/lib/pool \
290         /usr/lib/python2.6 \
291         /usr/lib/python2.6/vendor-packages \
292         /usr/lib/python2.6/vendor-packages/64 \
293         /usr/lib/python2.6/vendor-packages/solaris \
294         /usr/lib/python2.6/vendor-packages/zfs \
295         /usr/lib/python2.6/vendor-packages/beadm \
296         /usr/lib/rcap \
297         /usr/lib/rcap/$(MACH32) \
298         /usr/lib/sa  \
299         /usr/lib/saf \
300         /usr/lib/sasl \
301         /usr/lib/scsi \
302         /usr/lib/secure \
303         /usr/lib/security \
304         /usr/lib/smbsrv \
305         /usr/lib/vscan \
306         /usr/lib/zfs \
307         /usr/lib/zones \
308         /usr/old \
309         /usr/platform  \
310         /usr/proc \
311         /usr/proc/bin \
312         /usr/sadm \
313         /usr/sadm/install \
314         /usr/sadm/install/bin \
315         /usr/sadm/install/scripts \
316         /usr/sbin \
317         /usr/sbin/$(MACH32) \
318         /usr/share  \
319         /usr/share/applications \
320         /usr/share/audio \
321         /usr/share/audio/samples \
322         /usr/share/audio/samples/au \
323         /usr/share/gnome \
324         /usr/share/gnome/autostart \
325         /usr/share/hwdata \
326         /usr/share/lib \
```



```
327         /usr/share/lib/ccs \
328         /usr/share/lib/tmac  \
329         /usr/share/lib/ldif  \
330         /usr/share/lib/xml \
331         /usr/share/lib/xml/dtd \
332         /usr/share/man \
333         /usr/share/man/man1 \
334         /usr/share/man/man1b \
335         /usr/share/man/man1c \
336         /usr/share/man/man1m \
337         /usr/share/man/man2 \
338         /usr/share/man/man3 \
339         /usr/share/man/man3bsm \
340         /usr/share/man/man3c \
341         /usr/share/man/man3c_db \
342         /usr/share/man/man3cfgadm \
343         /usr/share/man/man3commputil \
344         /usr/share/man/man3contract \
345         /usr/share/man/man3cpc \
346         /usr/share/man/man3curses \
347         /usr/share/man/man3dat \
348         /usr/share/man/man3devid \
349         /usr/share/man/man3devinfo \
350         /usr/share/man/man3dlpi \
351         /usr/share/man/man3dns_sd \
352         /usr/share/man/man3elf \
353         /usr/share/man/man3exacct \
354         /usr/share/man/man3ext \
355         /usr/share/man/man3fcoe \
356         /usr/share/man/man3fstyp \
357         /usr/share/man/man3gen \
358         /usr/share/man/man3gss \
359         /usr/share/man/man3head \
360         /usr/share/man/man3iscsit \
361         /usr/share/man/man3kstat \
362         /usr/share/man/man3kvm \
363         /usr/share/man/man3ldap \
364         /usr/share/man/man3lgrp \
365         /usr/share/man/man3lib \
366         /usr/share/man/man3m \
367 #endif /* ! codereview */
368         /usr/share/man/man3mail \
369         /usr/share/man/man3malloc \
370         /usr/share/man/man3mp \
371         /usr/share/man/man3mpapi \
372         /usr/share/man/man3mvec \
373 #endif /* ! codereview */
374         /usr/share/man/man3nsl \
375         /usr/share/man/man3nvpair \
376         /usr/share/man/man3pam \
377         /usr/share/man/man3papi \
378         /usr/share/man/man3perl \
379         /usr/share/man/man3picl \
380         /usr/share/man/man3picltree \
381         /usr/share/man/man3pool \
382         /usr/share/man/man3proc \
383         /usr/share/man/man3project \
384         /usr/share/man/man3resolv \
385         /usr/share/man/man3rpc \
386         /usr/share/man/man3rsm \
387         /usr/share/man/man3sasl \
388         /usr/share/man/man3scf \
389         /usr/share/man/man3sec \
390         /usr/share/man/man3secdb \
391         /usr/share/man/man3sip \
392         /usr/share/man/man3slp \
```

```
393         /usr/share/man/man3socket \
394         /usr/share/man/man3stmf \
395         /usr/share/man/man3sysevent \
396         /usr/share/man/man3tecla \
397         /usr/share/man/man3tnf \
398         /usr/share/man/man3tsol \
399         /usr/share/man/man3uuid \
400         /usr/share/man/man3volmgt \
401         /usr/share/man/man3xcurses \
402         /usr/share/man/man3xnet \
403         /usr/share/man/man4 \
404         /usr/share/man/man5 \
405         /usr/share/man/man7 \
406         /usr/share/man/man7d \
407         /usr/share/man/man7fs \
408         /usr/share/man/man7i \
409         /usr/share/man/man7ipp \
410         /usr/share/man/man7m \
411         /usr/share/man/man7p \
412         /usr/share/man/man9 \
413         /usr/share/man/man9e \
414         /usr/share/man/man9f \
415         /usr/share/man/man9p \
416         /usr/share/man/man9s \
417         /usr/share/src \
418         /usr/snadm \
419         /usr/snadm/lib \
420         /usr/ucb \
421         $(MACH32_DIRS) \
422         /usr/ucblib \
423         /usr/xpg4 \
424         /usr/xpg4/bin \
425         /usr/xpg4/include \
426         /usr/xpg4/lib \
427         /usr/xpg6 \
428         /usr/xpg6/bin \
429         /var  \
430         /var/adm  \
431         /var/adm/exacct \
432         /var/adm/log \
433         /var/adm/pool \
434         /var/adm/sa \
435         /var/adm/sm.bin \
436         /var/adm/streams \
437         /var/cores \
438         /var/cron \
439         /var/db \
440         /var/db/ipf \
441         /var/games \
442         /var/idmap \
443         /var/krb5 \
444         /var/krb5/rcache  \
445         /var/krb5/rcache/root  \
446         /var/ld \
447         /var/log \
448         /var/log/pool \
449         /var/logadm \
450         /var/mail \
451         /var/news \
452         /var/opt \
453         /var/preserve \
454         /var/run \
455         /var/saf \
456         /var/sadm \
457         /var/sadm/install \
458         /var/sadm/install/admin \
```



```
459         /var/sadm/install/logs \
460         /var/sadm/pkg \
461         /var/sadm/security \
462         /var/smb \
463         /var/smb/cvol \
464         /var/smb/cvol/windows \
465         /var/smb/cvol/windows/system32 \
466         /var/smb/cvol/windows/system32/vss \
467         /var/spool \
468         /var/spool/cron \
469         /var/spool/cron/atjobs \
470         /var/spool/cron/crontabs \
471         /var/spool/lp \
472         /var/spool/pkg \
473         /var/spool/uucp \
474         /var/spool/uucppublic \
475         /var/svc \
476         /var/svc/log \
477         /var/svc/manifest \
478         /var/svc/manifest/milestone \
479         /var/svc/manifest/device \
480         /var/svc/manifest/system \
481         /var/svc/manifest/system/device \
482         /var/svc/manifest/system/filesystem \
483         /var/svc/manifest/system/security \
484         /var/svc/manifest/system/svc \
485         /var/svc/manifest/network \
486         /var/svc/manifest/network/dns \
487         /var/svc/manifest/network/ipsec \
488         /var/svc/manifest/network/ldap \
489         /var/svc/manifest/network/nfs \
490         /var/svc/manifest/network/nis \
491         /var/svc/manifest/network/rpc \
492         /var/svc/manifest/network/routing \
493         /var/svc/manifest/network/security \
494         /var/svc/manifest/network/shares \
495         /var/svc/manifest/network/ssl \
496         /var/svc/manifest/application \
497         /var/svc/manifest/application/management \
498         /var/svc/manifest/application/print \
499         /var/svc/manifest/application/security \
500         /var/svc/manifest/platform \
501         /var/svc/manifest/platform/sun4u \
502         /var/svc/manifest/platform/sun4v \
503         /var/svc/manifest/site \
504         /var/svc/profile \
505         /var/uucp \
506         /var/tmp \
507         /var/tsol \
508         /var/tsol/doors

510 sparcv9_DIRS64= \
511         /platform/sun4u \
512         /platform/sun4u/lib \
513         /platform/sun4u/lib/$(MACH64) \
514         /usr/platform/sun4u \
515         /usr/platform/sun4u/sbin \
516         /usr/platform/sun4u/lib \
517         /platform/sun4v/lib \
518         /platform/sun4v/lib/$(MACH64) \
519         /usr/platform/sun4v/sbin \
520         /usr/platform/sun4v/lib \
521         /usr/platform/sun4u-us3/lib \
522         /usr/platform/sun4u-opl/lib

524 amd64_DIRS64= \
```

```
525         /platform/i86pc/amd64

527 DIRS64= \
528         $($(MACH64)_DIRS64) \
529         /lib/$(MACH64) \
530         /lib/crypto/$(MACH64) \
531         /lib/fm/$(MACH64) \
532         /lib/secure/$(MACH64) \
533         /usr/bin/$(MACH64) \
534         /usr/ccs/bin/$(MACH64)  \
535         /usr/ccs/lib/$(MACH64) \
536         /usr/lib/$(MACH64) \
537         /usr/lib/$(MACH64)/gss \
538         /usr/lib/brand/sn1/$(MACH64) \
539         /usr/lib/brand/solaris10/$(MACH64) \
540         /usr/lib/elfedit/$(MACH64) \
541         /usr/lib/fm/$(MACH64) \
542         /usr/lib/fs/nfs/$(MACH64) \
543         /usr/lib/fs/smb/$(MACH64) \
544         /usr/lib/inet/$(MACH64) \
545         /usr/lib/krb5/$(MACH64) \
546         /usr/lib/libp/$(MACH64) \
547         /usr/lib/link_audit/$(MACH64) \
548         /usr/lib/lwp/$(MACH64) \
549         /usr/lib/mdb/kvm/$(MACH64) \
550         /usr/lib/mdb/proc/$(MACH64) \
551         /usr/lib/rcap/$(MACH64) \
552         /usr/lib/sasl/$(MACH64) \
553         /usr/lib/scsi/$(MACH64) \
554         /usr/lib/secure/$(MACH64) \
555         /usr/lib/security/$(MACH64) \
556         /usr/lib/smbsrv/$(MACH64) \
557         /usr/lib/abi/$(MACH64) \
558         /usr/sbin/$(MACH64) \
559         /usr/ucb/$(MACH64) \
560         /usr/ucblib/$(MACH64) \
561         /usr/xpg4/lib/$(MACH64) \
562         /var/ld/$(MACH64)

564 # /var/mail/:saved is built directly by the rootdirs target in
565 # /usr/src/Makefile because of the colon in its name.

567 # macros for symbolic links
568 SYM.DIRS= \
569         /bin \
570         /dev/stdin \
571         /dev/stdout \
572         /dev/stderr \
573         /etc/lib/ld.so.1 \
574         /etc/lib/libdl.so.1 \
575         /etc/lib/nss_files.so.1 \
576         /etc/log \
577         /lib/32 \
578         /lib/crypto/32 \
579         /lib/secure/32 \
580         /usr/adm \
581         /usr/spool \
582         /usr/lib/tmac \
583         /usr/ccs/lib/link_audit \
584         /usr/news \
585         /usr/preserve \
586         /usr/lib/32 \
587         /usr/lib/cron \
588         /usr/lib/elfedit/32 \
589         /usr/lib/libp/32 \
590         /usr/lib/lwp/32 \
```



```
591         /usr/lib/link_audit/32 \
592         /usr/lib/secure/32 \
593         /usr/mail \
594         /usr/man \
595         /usr/pub \
596         /usr/src \
597         /usr/tmp \
598         /usr/ucblib/32 \
599         /var/ld/32

601 sparc_SYM.DIRS64=

603 SYM.DIRS64= \
604         $($(MACH)_SYM.DIRS64) \
605         /lib/64 \
606         /lib/crypto/64 \
607         /lib/secure/64 \
608         /usr/lib/64 \
609         /usr/lib/brand/sn1/64 \
610         /usr/lib/brand/solaris10/64 \
611         /usr/lib/elfedit/64 \
612         /usr/lib/libp/64 \
613         /usr/lib/link_audit/64 \
614         /usr/lib/lwp/64 \
615         /usr/lib/secure/64 \
616         /usr/lib/security/64 \
617         /usr/xpg4/lib/64 \
618         /var/ld/64 \
619         /usr/ucblib/64

621 # prepend the ROOT prefix

623 ROOTDIRS=       $(TARGETDIRS:%=$(ROOT)%)

625 # conditional assignments
626 #
627 # Target directories with non-default values for owner and group must
628 # be referenced here, using their fully-prefixed names, and the non-
629 # default values assigned.  If a directory is mentioned above and not
630 # mentioned below, it has default values for attributes.
631 #
632 # The default value for DIRMODE is specified in usr/src/Makefile.master.
633 #

635 $(ROOT)/var/adm \
636 $(ROOT)/var/adm/sa :=           DIRMODE= 775

638 $(ROOT)/var/spool/lp:=          DIRMODE= 775

640 # file mode
641 #
642 $(ROOT)/tmp \
643 $(ROOT)/var/krb5/rcache \
644 $(ROOT)/var/preserve \
645 $(ROOT)/var/spool/pkg \
646 $(ROOT)/var/spool/uucppublic \
647 $(ROOT)/var/tmp:=       DIRMODE= 1777

649 $(ROOT)/root:=          DIRMODE= 700

651 $(ROOT)/var/krb5/rcache/root:=  DIRMODE= 700


654 #
655 # These permissions must match those set
656 # in the package manifests.
```

```
#
$(ROOT)/var/sadm/pkg \
$(ROOT)/var/sadm/security \
$(ROOT)/var/sadm/install/logs :=        DIRMODE= 555


#
# These permissions must match the ones set
# internally by fdfs and autofs.
#
$(ROOT)/dev/fd \
$(ROOT)/home:=          DIRMODE= 555

$(ROOT)/var/mail:=      DIRMODE=1777

$(ROOT)/proc:=          DIRMODE= 555

$(ROOT)/system/contract:=       DIRMODE= 555
$(ROOT)/system/object:=         DIRMODE= 555

# symlink assignments, LINKDEST is the value of the symlink
#
$(ROOT)/usr/lib/cron:=                  LINKDEST=../../etc/cron.d
$(ROOT)/bin:=                           LINKDEST=usr/bin
$(ROOT)/lib/32:=                        LINKDEST=.
$(ROOT)/lib/crypto/32:=                 LINKDEST=.
$(ROOT)/lib/secure/32:=                 LINKDEST=.
$(ROOT)/dev/stdin:=                     LINKDEST=fd/0
$(ROOT)/dev/stdout:=                    LINKDEST=fd/1
$(ROOT)/dev/stderr:=                    LINKDEST=fd/2
$(ROOT)/usr/pub:=                       LINKDEST=share/lib/pub
$(ROOT)/usr/man:=                       LINKDEST=share/man
$(ROOT)/usr/src:=                       LINKDEST=share/src
$(ROOT)/usr/adm:=                       LINKDEST=../var/adm
$(ROOT)/etc/lib/ld.so.1:=               LINKDEST=../../lib/ld.so.1
$(ROOT)/etc/lib/libdl.so.1:=            LINKDEST=../../lib/libdl.so.1
$(ROOT)/etc/lib/nss_files.so.1:=        LINKDEST=../../lib/nss_files.so.1
$(ROOT)/etc/log:=                       LINKDEST=../var/adm/log
$(ROOT)/usr/mail:=                      LINKDEST=../var/mail
$(ROOT)/usr/news:=                      LINKDEST=../var/news
$(ROOT)/usr/preserve:=                  LINKDEST=../var/preserve
$(ROOT)/usr/spool:=                     LINKDEST=../var/spool
$(ROOT)/usr/tmp:=                       LINKDEST=../var/tmp
$(ROOT)/usr/lib/tmac:=                  LINKDEST=../share/lib/tmac
$(ROOT)/usr/lib/32:=                    LINKDEST=.
$(ROOT)/usr/lib/elfedit/32:=            LINKDEST=.
$(ROOT)/usr/lib/libp/32:=               LINKDEST=.
$(ROOT)/usr/lib/lwp/32:=                LINKDEST=.
$(ROOT)/usr/lib/link_audit/32:=         LINKDEST=.
$(ROOT)/usr/lib/secure/32:=             LINKDEST=.
$(ROOT)/usr/ccs/lib/link_audit:=        LINKDEST=../../lib/link_audit
$(ROOT)/var/ld/32:=                     LINKDEST=.
$(ROOT)/usr/ucblib/32:=                 LINKDEST=.


$(BUILD64) $(ROOT)/lib/64:=             LINKDEST=$(MACH64)
$(BUILD64) $(ROOT)/lib/crypto/64:=      LINKDEST=$(MACH64)
$(BUILD64) $(ROOT)/lib/secure/64:=      LINKDEST=$(MACH64)
$(BUILD64) $(ROOT)/usr/lib/64:=         LINKDEST=$(MACH64)
$(BUILD64) $(ROOT)/usr/lib/elfedit/64:= LINKDEST=$(MACH64)
$(BUILD64) $(ROOT)/usr/lib/brand/sn1/64:=       LINKDEST=$(MACH64)
$(BUILD64) $(ROOT)/usr/lib/brand/solaris10/64:= LINKDEST=$(MACH64)
$(BUILD64) $(ROOT)/usr/lib/libp/64:=    LINKDEST=$(MACH64)
$(BUILD64) $(ROOT)/usr/lib/lwp/64:=     LINKDEST=$(MACH64)
$(BUILD64) $(ROOT)/usr/lib/link_audit/64:=      LINKDEST=$(MACH64)
$(BUILD64) $(ROOT)/usr/lib/secure/64:=  LINKDEST=$(MACH64)
```



```
$(BUILD64) $(ROOT)/usr/lib/security/64:=        LINKDEST=$(MACH64)
$(BUILD64) $(ROOT)/usr/xpg4/lib/64:=    LINKDEST=$(MACH64)
$(BUILD64) $(ROOT)/var/ld/64:=          LINKDEST=$(MACH64)
$(BUILD64) $(ROOT)/usr/ucblib/64:=      LINKDEST=$(MACH64)

#
# Installing a directory symlink calls for overriding INS.dir to install
# a symlink.
#
$(DIRLINKS:%=$(ROOT)%):= \
    INS.dir= -$(RM) -r $@; $(SYMLINK) $(LINKDEST) $@

# Special symlinks to populate usr/ccs/lib, whose objects
# have actually been moved to usr/lib
# Rather than adding another set of rules, we add usr/lib/lwp files here
$(ROOT)/usr/ccs/lib/libcurses.so:=      REALPATH=../../../lib/libcurses.so.1
$(ROOT)/usr/ccs/lib/llib-lcurses:=      REALPATH=../../../lib/llib-lcurses
$(ROOT)/usr/ccs/lib/llib-lcurses.ln:=   REALPATH=../../../lib/llib-lcurses.ln
$(ROOT)/usr/ccs/lib/libform.so:=        REALPATH=../../lib/libform.so.1
$(ROOT)/usr/ccs/lib/llib-lform:=        REALPATH=../../lib/llib-lform
$(ROOT)/usr/ccs/lib/llib-lform.ln:=     REALPATH=../../lib/llib-lform.ln
$(ROOT)/usr/ccs/lib/libgen.so:=         REALPATH=../../../lib/libgen.so.1
$(ROOT)/usr/ccs/lib/llib-lgen:=         REALPATH=../../../lib/llib-lgen
$(ROOT)/usr/ccs/lib/llib-lgen.ln:=      REALPATH=../../../lib/llib-lgen.ln
$(ROOT)/usr/ccs/lib/libmalloc.so:=      REALPATH=../../lib/libmalloc.so.1
$(ROOT)/usr/ccs/lib/libmenu.so:=        REALPATH=../../lib/libmenu.so.1
$(ROOT)/usr/ccs/lib/llib-lmenu:=        REALPATH=../../lib/llib-lmenu
$(ROOT)/usr/ccs/lib/llib-lmenu.ln:=     REALPATH=../../lib/llib-lmenu.ln
$(ROOT)/usr/ccs/lib/libpanel.so:=       REALPATH=../../lib/libpanel.so.1
$(ROOT)/usr/ccs/lib/llib-lpanel:=       REALPATH=../../lib/llib-lpanel
$(ROOT)/usr/ccs/lib/llib-lpanel.ln:=    REALPATH=../../lib/llib-lpanel.ln
$(ROOT)/usr/ccs/lib/libtermlib.so:=     REALPATH=../../../lib/libcurses.so.1
$(ROOT)/usr/ccs/lib/llib-ltermlib:=     REALPATH=../../../lib/llib-lcurses
$(ROOT)/usr/ccs/lib/llib-ltermlib.ln:=  REALPATH=../../../lib/llib-lcurses.ln
$(ROOT)/usr/ccs/lib/libtermcap.so:=     REALPATH=../../../lib/libtermcap.so.1
$(ROOT)/usr/ccs/lib/llib-ltermcap:=     REALPATH=../../../lib/llib-ltermcap
$(ROOT)/usr/ccs/lib/llib-ltermcap.ln:=  REALPATH=../../../lib/llib-ltermcap.ln
$(ROOT)/usr/ccs/lib/values-Xa.o:=       REALPATH=../../lib/values-Xa.o
$(ROOT)/usr/ccs/lib/values-Xc.o:=       REALPATH=../../lib/values-Xc.o
$(ROOT)/usr/ccs/lib/values-Xs.o:=       REALPATH=../../lib/values-Xs.o
$(ROOT)/usr/ccs/lib/values-Xt.o:=       REALPATH=../../lib/values-Xt.o
$(ROOT)/usr/ccs/lib/values-xpg4.o:=     REALPATH=../../lib/values-xpg4.o
$(ROOT)/usr/ccs/lib/values-xpg6.o:=     REALPATH=../../lib/values-xpg6.o
$(ROOT)/usr/ccs/lib/libl.so:=           REALPATH=../../lib/libl.so.1
$(ROOT)/usr/ccs/lib/llib-ll.ln:=        REALPATH=../../lib/llib-ll.ln
$(ROOT)/usr/ccs/lib/liby.so:=           REALPATH=../../lib/liby.so.1
$(ROOT)/usr/ccs/lib/llib-ly.ln:=        REALPATH=../../lib/llib-ly.ln
$(ROOT)/usr/lib/libp/libc.so.1:=        REALPATH=../../../lib/libc.so.1
$(ROOT)/usr/lib/lwp/libthread.so.1:=    REALPATH=../libthread.so.1
$(ROOT)/usr/lib/lwp/libthread_db.so.1:= REALPATH=../libthread_db.so.1

# symlinks to populate usr/ccs/lib/$(MACH64)
$(ROOT)/usr/ccs/lib/$(MACH64)/libcurses.so:= \
        REALPATH=../../../../lib/$(MACH64)/libcurses.so.1
$(ROOT)/usr/ccs/lib/$(MACH64)/llib-lcurses.ln:= \
        REALPATH=../../../../lib/$(MACH64)/llib-lcurses.ln
$(ROOT)/usr/ccs/lib/$(MACH64)/libform.so:= \
        REALPATH=../../../lib/$(MACH64)/libform.so.1
$(ROOT)/usr/ccs/lib/$(MACH64)/llib-lform.ln:= \
        REALPATH=../../../lib/$(MACH64)/llib-lform.ln
$(ROOT)/usr/ccs/lib/$(MACH64)/libgen.so:= \
        REALPATH=../../../../lib/$(MACH64)/libgen.so.1
$(ROOT)/usr/ccs/lib/$(MACH64)/llib-lgen.ln:= \
        REALPATH=../../../../lib/$(MACH64)/llib-lgen.ln
$(ROOT)/usr/ccs/lib/$(MACH64)/libmalloc.so:= \
        REALPATH=../../../lib/$(MACH64)/libmalloc.so.1
```

```
$(ROOT)/usr/ccs/lib/$(MACH64)/libmenu.so:= \
        REALPATH=../../../lib/$(MACH64)/libmenu.so.1
$(ROOT)/usr/ccs/lib/$(MACH64)/llib-lmenu.ln:= \
        REALPATH=../../../lib/$(MACH64)/llib-lmenu.ln
$(ROOT)/usr/ccs/lib/$(MACH64)/libpanel.so:= \
        REALPATH=../../../lib/$(MACH64)/libpanel.so.1
$(ROOT)/usr/ccs/lib/$(MACH64)/llib-lpanel.ln:= \
        REALPATH=../../../lib/$(MACH64)/llib-lpanel.ln
$(ROOT)/usr/ccs/lib/$(MACH64)/libtermlib.so:= \
        REALPATH=../../../../lib/$(MACH64)/libcurses.so.1
$(ROOT)/usr/ccs/lib/$(MACH64)/llib-ltermlib.ln:= \
        REALPATH=../../../../lib/$(MACH64)/llib-lcurses.ln
$(ROOT)/usr/ccs/lib/$(MACH64)/libtermcap.so:= \
        REALPATH=../../../../lib/$(MACH64)/libtermcap.so.1
$(ROOT)/usr/ccs/lib/$(MACH64)/llib-ltermcap.ln:= \
        REALPATH=../../../../lib/$(MACH64)/llib-ltermcap.ln
$(ROOT)/usr/ccs/lib/$(MACH64)/values-Xa.o:= \
        REALPATH=../../../lib/$(MACH64)/values-Xa.o
$(ROOT)/usr/ccs/lib/$(MACH64)/values-Xc.o:= \
        REALPATH=../../../lib/$(MACH64)/values-Xc.o
$(ROOT)/usr/ccs/lib/$(MACH64)/values-Xs.o:= \
        REALPATH=../../../lib/$(MACH64)/values-Xs.o
$(ROOT)/usr/ccs/lib/$(MACH64)/values-Xt.o:= \
        REALPATH=../../../lib/$(MACH64)/values-Xt.o
$(ROOT)/usr/ccs/lib/$(MACH64)/values-xpg4.o:= \
        REALPATH=../../../lib/$(MACH64)/values-xpg4.o
$(ROOT)/usr/ccs/lib/$(MACH64)/values-xpg6.o:= \
        REALPATH=../../../lib/$(MACH64)/values-xpg6.o
$(ROOT)/usr/ccs/lib/$(MACH64)/libl.so:= \
        REALPATH=../../../lib/$(MACH64)/libl.so.1
$(ROOT)/usr/ccs/lib/$(MACH64)/llib-ll.ln:= \
        REALPATH=../../../lib/$(MACH64)/llib-ll.ln
$(ROOT)/usr/ccs/lib/$(MACH64)/liby.so:= \
        REALPATH=../../../lib/$(MACH64)/liby.so.1
$(ROOT)/usr/ccs/lib/$(MACH64)/llib-ly.ln:= \
        REALPATH=../../../lib/$(MACH64)/llib-ly.ln
$(ROOT)/usr/lib/libp/$(MACH64)/libc.so.1:= \
        REALPATH=../../../../lib/$(MACH64)/libc.so.1
$(ROOT)/usr/lib/lwp/$(MACH64)/libthread.so.1:= \
        REALPATH=../../$(MACH64)/libthread.so.1
$(ROOT)/usr/lib/lwp/$(MACH64)/libthread_db.so.1:= \
        REALPATH=../../$(MACH64)/libthread_db.so.1

SYM.USRCCSLIB= \
        /usr/ccs/lib/libcurses.so \
        /usr/ccs/lib/llib-lcurses \
        /usr/ccs/lib/llib-lcurses.ln \
        /usr/ccs/lib/libform.so \
        /usr/ccs/lib/llib-lform \
        /usr/ccs/lib/llib-lform.ln \
        /usr/ccs/lib/libgen.so \
        /usr/ccs/lib/llib-lgen \
        /usr/ccs/lib/llib-lgen.ln \
        /usr/ccs/lib/libmalloc.so \
        /usr/ccs/lib/libmenu.so \
        /usr/ccs/lib/llib-lmenu \
        /usr/ccs/lib/llib-lmenu.ln \
        /usr/ccs/lib/libpanel.so \
        /usr/ccs/lib/llib-lpanel \
        /usr/ccs/lib/llib-lpanel.ln \
        /usr/ccs/lib/libtermlib.so \
        /usr/ccs/lib/llib-ltermlib \
        /usr/ccs/lib/llib-ltermlib.ln \
        /usr/ccs/lib/libtermcap.so \
        /usr/ccs/lib/llib-ltermcap \
        /usr/ccs/lib/llib-ltermcap.ln \
```



```
        /usr/ccs/lib/values-Xa.o \
        /usr/ccs/lib/values-Xc.o \
        /usr/ccs/lib/values-Xs.o \
        /usr/ccs/lib/values-Xt.o \
        /usr/ccs/lib/values-xpg4.o \
        /usr/ccs/lib/values-xpg6.o \
        /usr/ccs/lib/libl.so \
        /usr/ccs/lib/llib-ll.ln \
        /usr/ccs/lib/liby.so \
        /usr/ccs/lib/llib-ly.ln \
        /usr/lib/libp/libc.so.1 \
        /usr/lib/lwp/libthread.so.1 \
        /usr/lib/lwp/libthread_db.so.1

SYM.USRCCSLIB64= \
        /usr/ccs/lib/$(MACH64)/libcurses.so \
        /usr/ccs/lib/$(MACH64)/llib-lcurses.ln \
        /usr/ccs/lib/$(MACH64)/libform.so \
        /usr/ccs/lib/$(MACH64)/llib-lform.ln \
        /usr/ccs/lib/$(MACH64)/libgen.so \
        /usr/ccs/lib/$(MACH64)/llib-lgen.ln \
        /usr/ccs/lib/$(MACH64)/libmalloc.so \
        /usr/ccs/lib/$(MACH64)/libmenu.so \
        /usr/ccs/lib/$(MACH64)/llib-lmenu.ln \
        /usr/ccs/lib/$(MACH64)/libpanel.so \
        /usr/ccs/lib/$(MACH64)/llib-lpanel.ln \
        /usr/ccs/lib/$(MACH64)/libtermlib.so \
        /usr/ccs/lib/$(MACH64)/llib-ltermlib.ln \
        /usr/ccs/lib/$(MACH64)/libtermcap.so \
        /usr/ccs/lib/$(MACH64)/llib-ltermcap.ln \
        /usr/ccs/lib/$(MACH64)/values-Xa.o \
        /usr/ccs/lib/$(MACH64)/values-Xc.o \
        /usr/ccs/lib/$(MACH64)/values-Xs.o \
        /usr/ccs/lib/$(MACH64)/values-Xt.o \
        /usr/ccs/lib/$(MACH64)/values-xpg4.o \
        /usr/ccs/lib/$(MACH64)/values-xpg6.o \
        /usr/ccs/lib/$(MACH64)/libl.so \
        /usr/ccs/lib/$(MACH64)/llib-ll.ln \
        /usr/ccs/lib/$(MACH64)/liby.so \
        /usr/ccs/lib/$(MACH64)/llib-ly.ln \
        /usr/lib/libp/$(MACH64)/libc.so.1 \
        /usr/lib/lwp/$(MACH64)/libthread.so.1 \
        /usr/lib/lwp/$(MACH64)/libthread_db.so.1

# Special symlinks to direct libraries that have been moved
# from /usr/lib to /lib in order to live in the root filesystem.
$(ROOT)/lib/libposix4.so.1:=            REALPATH=librt.so.1
$(ROOT)/lib/libposix4.so:=              REALPATH=libposix4.so.1
$(ROOT)/lib/llib-lposix4:=              REALPATH=llib-lrt
$(ROOT)/lib/llib-lposix4.ln:=           REALPATH=llib-lrt.ln
$(ROOT)/lib/libthread_db.so.1:=         REALPATH=libc_db.so.1
$(ROOT)/lib/libthread_db.so:=           REALPATH=libc_db.so.1
$(ROOT)/usr/lib/ld.so.1:=               REALPATH=../../lib/ld.so.1
$(ROOT)/usr/lib/libadm.so.1:=           REALPATH=../../lib/libadm.so.1
$(ROOT)/usr/lib/libadm.so:=             REALPATH=../../lib/libadm.so.1
$(ROOT)/usr/lib/libaio.so.1:=           REALPATH=../../lib/libaio.so.1
$(ROOT)/usr/lib/libaio.so:=             REALPATH=../../lib/libaio.so.1
$(ROOT)/usr/lib/libavl.so.1:=           REALPATH=../../lib/libavl.so.1
$(ROOT)/usr/lib/libavl.so:=             REALPATH=../../lib/libavl.so.1
$(ROOT)/usr/lib/libbsm.so.1:=           REALPATH=../../lib/libbsm.so.1
$(ROOT)/usr/lib/libbsm.so:=             REALPATH=../../lib/libbsm.so.1
$(ROOT)/usr/lib/libc.so.1:=             REALPATH=../../lib/libc.so.1
$(ROOT)/usr/lib/libc.so:=               REALPATH=../../lib/libc.so.1
$(ROOT)/usr/lib/libc_db.so.1:=          REALPATH=../../lib/libc_db.so.1
$(ROOT)/usr/lib/libc_db.so:=            REALPATH=../../lib/libc_db.so.1
$(ROOT)/usr/lib/libcmdutils.so.1:=      REALPATH=../../lib/libcmdutils.so.1
```

```
$(ROOT)/usr/lib/libcmdutils.so:=          REALPATH=../../lib/libcmdutils.so.1
$(ROOT)/usr/lib/libcontract.so.1:=        REALPATH=../../lib/libcontract.so.1
$(ROOT)/usr/lib/libcontract.so:=          REALPATH=../../lib/libcontract.so.1
$(ROOT)/usr/lib/libcryptoutil.so.1:=      REALPATH=../../lib/libcryptoutil.so.1
$(ROOT)/usr/lib/libcryptoutil.so:=        REALPATH=../../lib/libcryptoutil.so.1
$(ROOT)/usr/lib/libctf.so.1:=             REALPATH=../../lib/libctf.so.1
$(ROOT)/usr/lib/libctf.so:=               REALPATH=../../lib/libctf.so.1
$(ROOT)/usr/lib/libcurses.so.1:=          REALPATH=../../lib/libcurses.so.1
$(ROOT)/usr/lib/libcurses.so:=            REALPATH=../../lib/libcurses.so.1
$(ROOT)/usr/lib/libdevice.so.1:=          REALPATH=../../lib/libdevice.so.1
$(ROOT)/usr/lib/libdevice.so:=            REALPATH=../../lib/libdevice.so.1
$(ROOT)/usr/lib/libdevid.so.1:=           REALPATH=../../lib/libdevid.so.1
$(ROOT)/usr/lib/libdevid.so:=             REALPATH=../../lib/libdevid.so.1
$(ROOT)/usr/lib/libdevinfo.so.1:=         REALPATH=../../lib/libdevinfo.so.1
$(ROOT)/usr/lib/libdevinfo.so:=           REALPATH=../../lib/libdevinfo.so.1
$(ROOT)/usr/lib/libdhcpagent.so.1:=       REALPATH=../../lib/libdhcpagent.so.1
$(ROOT)/usr/lib/libdhcpagent.so:=         REALPATH=../../lib/libdhcpagent.so.1
$(ROOT)/usr/lib/libdhcputil.so.1:=        REALPATH=../../lib/libdhcputil.so.1
$(ROOT)/usr/lib/libdhcputil.so:=          REALPATH=../../lib/libdhcputil.so.1
$(ROOT)/usr/lib/libdl.so.1:=              REALPATH=../../lib/libdl.so.1
$(ROOT)/usr/lib/libdl.so:=                REALPATH=../../lib/libdl.so.1
$(ROOT)/usr/lib/libdlpi.so.1:=            REALPATH=../../lib/libdlpi.so.1
$(ROOT)/usr/lib/libdlpi.so:=              REALPATH=../../lib/libdlpi.so.1
$(ROOT)/usr/lib/libdoor.so.1:=            REALPATH=../../lib/libdoor.so.1
$(ROOT)/usr/lib/libdoor.so:=              REALPATH=../../lib/libdoor.so.1
$(ROOT)/usr/lib/libefi.so.1:=             REALPATH=../../lib/libefi.so.1
$(ROOT)/usr/lib/libefi.so:=               REALPATH=../../lib/libefi.so.1
$(ROOT)/usr/lib/libelf.so.1:=             REALPATH=../../lib/libelf.so.1
$(ROOT)/usr/lib/libelf.so:=               REALPATH=../../lib/libelf.so.1
$(ROOT)/usr/lib/libfdisk.so.1:=           REALPATH=../../lib/libfdisk.so.1
$(ROOT)/usr/lib/libfdisk.so:=             REALPATH=../../lib/libfdisk.so.1
$(ROOT)/usr/lib/libgen.so.1:=             REALPATH=../../lib/libgen.so.1
$(ROOT)/usr/lib/libgen.so:=               REALPATH=../../lib/libgen.so.1
$(ROOT)/usr/lib/libinetutil.so.1:=        REALPATH=../../lib/libinetutil.so.1
$(ROOT)/usr/lib/libinetutil.so:=          REALPATH=../../lib/libinetutil.so.1
$(ROOT)/usr/lib/libintl.so.1:=            REALPATH=../../lib/libintl.so.1
$(ROOT)/usr/lib/libintl.so:=              REALPATH=../../lib/libintl.so.1
$(ROOT)/usr/lib/libkmf.so.1:=             REALPATH=../../lib/libkmf.so.1
$(ROOT)/usr/lib/libkmf.so:=               REALPATH=../../lib/libkmf.so.1
$(ROOT)/usr/lib/libkmfberder.so.1:=       REALPATH=../../lib/libkmfberder.so.1
$(ROOT)/usr/lib/libkmfberder.so:=         REALPATH=../../lib/libkmfberder.so.1
$(ROOT)/usr/lib/libkstat.so.1:=           REALPATH=../../lib/libkstat.so.1
$(ROOT)/usr/lib/libkstat.so:=             REALPATH=../../lib/libkstat.so.1
$(ROOT)/usr/lib/liblddbg.so.4:=           REALPATH=../../lib/liblddbg.so.4
$(ROOT)/usr/lib/libm.so.1:=               REALPATH=../../lib/libm.so.1
$(ROOT)/usr/lib/libm.so.2:=               REALPATH=../../lib/libm.so.2
$(ROOT)/usr/lib/libm.so:=                 REALPATH=../../lib/libm.so.2
$(ROOT)/usr/lib/libmd.so.1:=              REALPATH=../../lib/libmd.so.1
$(ROOT)/usr/lib/libmd.so:=                REALPATH=../../lib/libmd.so.1
$(ROOT)/usr/lib/libmd5.so.1:=             REALPATH=../../lib/libmd5.so.1
$(ROOT)/usr/lib/libmd5.so:=               REALPATH=../../lib/libmd5.so.1
$(ROOT)/usr/lib/libmeta.so.1:=            REALPATH=../../lib/libmeta.so.1
$(ROOT)/usr/lib/libmeta.so:=              REALPATH=../../lib/libmeta.so.1
$(ROOT)/usr/lib/libmp.so.1:=              REALPATH=../../lib/libmp.so.1
$(ROOT)/usr/lib/libmp.so.2:=              REALPATH=../../lib/libmp.so.2
$(ROOT)/usr/lib/libmp.so:=                REALPATH=../../lib/libmp.so.2
$(ROOT)/usr/lib/libmvec.so.1:=            REALPATH=../../lib/libmvec.so.1
$(ROOT)/usr/lib/libmvec.so:=              REALPATH=../../lib/libmvec.so.1
$(ROOT)/usr/lib/libnsl.so.1:=             REALPATH=../../lib/libnsl.so.1
$(ROOT)/usr/lib/libnsl.so:=               REALPATH=../../lib/libnsl.so.1
$(ROOT)/usr/lib/libnvpair.so.1:=          REALPATH=../../lib/libnvpair.so.1
$(ROOT)/usr/lib/libnvpair.so:=            REALPATH=../../lib/libnvpair.so.1
$(ROOT)/usr/lib/libpam.so.1:=             REALPATH=../../lib/libpam.so.1
$(ROOT)/usr/lib/libpam.so:=               REALPATH=../../lib/libpam.so.1
$(ROOT)/usr/lib/libposix4.so.1:=          REALPATH=../../lib/librt.so.1
$(ROOT)/usr/lib/libposix4.so:=            REALPATH=../../lib/librt.so.1
```



```
$(ROOT)/usr/lib/libproc.so.1:=            REALPATH=../../lib/libproc.so.1
$(ROOT)/usr/lib/libproc.so:=              REALPATH=../../lib/libproc.so.1
$(ROOT)/usr/lib/libpthread.so.1:=         REALPATH=../../lib/libpthread.so.1
$(ROOT)/usr/lib/libpthread.so:=           REALPATH=../../lib/libpthread.so.1
$(ROOT)/usr/lib/librcm.so.1:=             REALPATH=../../lib/librcm.so.1
$(ROOT)/usr/lib/librcm.so:=               REALPATH=../../lib/librcm.so.1
$(ROOT)/usr/lib/libresolv.so.1:=          REALPATH=../../lib/libresolv.so.1
$(ROOT)/usr/lib/libresolv.so.2:=          REALPATH=../../lib/libresolv.so.2
$(ROOT)/usr/lib/libresolv.so:=            REALPATH=../../lib/libresolv.so.2
$(ROOT)/usr/lib/librestart.so.1:=         REALPATH=../../lib/librestart.so.1
$(ROOT)/usr/lib/librestart.so:=           REALPATH=../../lib/librestart.so.1
$(ROOT)/usr/lib/librpcsvc.so.1:=          REALPATH=../../lib/librpcsvc.so.1
$(ROOT)/usr/lib/librpcsvc.so:=            REALPATH=../../lib/librpcsvc.so.1
$(ROOT)/usr/lib/librt.so.1:=              REALPATH=../../lib/librt.so.1
$(ROOT)/usr/lib/librt.so:=                REALPATH=../../lib/librt.so.1
$(ROOT)/usr/lib/librtld.so.1:=            REALPATH=../../lib/librtld.so.1
$(ROOT)/usr/lib/librtld_db.so.1:=         REALPATH=../../lib/librtld_db.so.1
$(ROOT)/usr/lib/librtld_db.so:=           REALPATH=../../lib/librtld_db.so.1
$(ROOT)/usr/lib/libscf.so.1:=             REALPATH=../../lib/libscf.so.1
$(ROOT)/usr/lib/libscf.so:=               REALPATH=../../lib/libscf.so.1
$(ROOT)/usr/lib/libsec.so.1:=             REALPATH=../../lib/libsec.so.1
$(ROOT)/usr/lib/libsec.so:=               REALPATH=../../lib/libsec.so.1
$(ROOT)/usr/lib/libsecdb.so.1:=           REALPATH=../../lib/libsecdb.so.1
$(ROOT)/usr/lib/libsecdb.so:=             REALPATH=../../lib/libsecdb.so.1
$(ROOT)/usr/lib/libsendfile.so.1:=        REALPATH=../../lib/libsendfile.so.1
$(ROOT)/usr/lib/libsendfile.so:=          REALPATH=../../lib/libsendfile.so.1
$(ROOT)/usr/lib/libsocket.so.1:=          REALPATH=../../lib/libsocket.so.1
$(ROOT)/usr/lib/libsocket.so:=            REALPATH=../../lib/libsocket.so.1
$(ROOT)/usr/lib/libsysevent.so.1:=        REALPATH=../../lib/libsysevent.so.1
$(ROOT)/usr/lib/libsysevent.so:=          REALPATH=../../lib/libsysevent.so.1
$(ROOT)/usr/lib/libtermcap.so.1:=         REALPATH=../../lib/libtermcap.so.1
$(ROOT)/usr/lib/libtermcap.so:=           REALPATH=../../lib/libtermcap.so.1
$(ROOT)/usr/lib/libtermlib.so.1:=         REALPATH=../../lib/libcurses.so.1
$(ROOT)/usr/lib/libtermlib.so:=           REALPATH=../../lib/libcurses.so.1
$(ROOT)/usr/lib/libthread.so.1:=          REALPATH=../../lib/libthread.so.1
$(ROOT)/usr/lib/libthread.so:=            REALPATH=../../lib/libthread.so.1
$(ROOT)/usr/lib/libthread_db.so.1:=       REALPATH=../../lib/libc_db.so.1
$(ROOT)/usr/lib/libthread_db.so:=         REALPATH=../../lib/libc_db.so.1
$(ROOT)/usr/lib/libtsnet.so.1:=           REALPATH=../../lib/libtsnet.so.1
$(ROOT)/usr/lib/libtsnet.so:=             REALPATH=../../lib/libtsnet.so.1
$(ROOT)/usr/lib/libtsol.so.2:=            REALPATH=../../lib/libtsol.so.2
$(ROOT)/usr/lib/libtsol.so:=              REALPATH=../../lib/libtsol.so.2
$(ROOT)/usr/lib/libumem.so.1:=            REALPATH=../../lib/libumem.so.1
$(ROOT)/usr/lib/libumem.so:=              REALPATH=../../lib/libumem.so.1
$(ROOT)/usr/lib/libuuid.so.1:=            REALPATH=../../lib/libuuid.so.1
$(ROOT)/usr/lib/libuuid.so:=              REALPATH=../../lib/libuuid.so.1
$(ROOT)/usr/lib/libuutil.so.1:=           REALPATH=../../lib/libuutil.so.1
$(ROOT)/usr/lib/libuutil.so:=             REALPATH=../../lib/libuutil.so.1
$(ROOT)/usr/lib/libw.so.1:=               REALPATH=../../lib/libw.so.1
$(ROOT)/usr/lib/libw.so:=                 REALPATH=../../lib/libw.so.1
$(ROOT)/usr/lib/libxnet.so.1:=            REALPATH=../../lib/libxnet.so.1
$(ROOT)/usr/lib/libxnet.so:=              REALPATH=../../lib/libxnet.so.1
$(ROOT)/usr/lib/libzfs.so.1:=             REALPATH=../../lib/libzfs.so.1
$(ROOT)/usr/lib/libzfs.so:=               REALPATH=../../lib/libzfs.so.1
$(ROOT)/usr/lib/libzfs_core.so.1:=        REALPATH=../../lib/libzfs_core.so.1
$(ROOT)/usr/lib/libzfs_core.so:=          REALPATH=../../lib/libzfs_core.so.1
$(ROOT)/usr/lib/llib-ladm.ln:=            REALPATH=../../lib/llib-ladm.ln
$(ROOT)/usr/lib/llib-ladm:=               REALPATH=../../lib/llib-ladm
$(ROOT)/usr/lib/llib-laio.ln:=            REALPATH=../../lib/llib-laio.ln
$(ROOT)/usr/lib/llib-laio:=               REALPATH=../../lib/llib-laio
$(ROOT)/usr/lib/llib-lavl.ln:=            REALPATH=../../lib/llib-lavl.ln
$(ROOT)/usr/lib/llib-lavl:=               REALPATH=../../lib/llib-lavl
$(ROOT)/usr/lib/llib-lbsm.ln:=            REALPATH=../../lib/llib-lbsm.ln
$(ROOT)/usr/lib/llib-lbsm:=               REALPATH=../../lib/llib-lbsm
$(ROOT)/usr/lib/llib-lc.ln:=              REALPATH=../../lib/llib-lc.ln
$(ROOT)/usr/lib/llib-lc:=                 REALPATH=../../lib/llib-lc
```

```
$(ROOT)/usr/lib/llib-lcmdutils.ln:=     REALPATH=../../lib/llib-lcmdutils.ln
$(ROOT)/usr/lib/llib-lcmdutils:=        REALPATH=../../lib/llib-lcmdutils
$(ROOT)/usr/lib/llib-lcontract.ln:=     REALPATH=../../lib/llib-lcontract.ln
$(ROOT)/usr/lib/llib-lcontract:=        REALPATH=../../lib/llib-lcontract
$(ROOT)/usr/lib/llib-lctf.ln:=          REALPATH=../../lib/llib-lctf.ln
$(ROOT)/usr/lib/llib-lctf:=             REALPATH=../../lib/llib-lctf
$(ROOT)/usr/lib/llib-lcurses.ln:=       REALPATH=../../lib/llib-lcurses.ln
$(ROOT)/usr/lib/llib-lcurses:=          REALPATH=../../lib/llib-lcurses
$(ROOT)/usr/lib/llib-ldevice.ln:=       REALPATH=../../lib/llib-ldevice.ln
$(ROOT)/usr/lib/llib-ldevice:=          REALPATH=../../lib/llib-ldevice
$(ROOT)/usr/lib/llib-ldevid.ln:=        REALPATH=../../lib/llib-ldevid.ln
$(ROOT)/usr/lib/llib-ldevid:=           REALPATH=../../lib/llib-ldevid
$(ROOT)/usr/lib/llib-ldevinfo.ln:=      REALPATH=../../lib/llib-ldevinfo.ln
$(ROOT)/usr/lib/llib-ldevinfo:=         REALPATH=../../lib/llib-ldevinfo
$(ROOT)/usr/lib/llib-ldhcpagent.ln:=    REALPATH=../../lib/llib-ldhcpagent.ln
$(ROOT)/usr/lib/llib-ldhcpagent:=       REALPATH=../../lib/llib-ldhcpagent
$(ROOT)/usr/lib/llib-ldhcputil.ln:=     REALPATH=../../lib/llib-ldhcputil.ln
$(ROOT)/usr/lib/llib-ldhcputil:=        REALPATH=../../lib/llib-ldhcputil
$(ROOT)/usr/lib/llib-ldl.ln:=           REALPATH=../../lib/llib-ldl.ln
$(ROOT)/usr/lib/llib-ldl:=              REALPATH=../../lib/llib-ldl
$(ROOT)/usr/lib/llib-ldoor.ln:=         REALPATH=../../lib/llib-ldoor.ln
$(ROOT)/usr/lib/llib-ldoor:=            REALPATH=../../lib/llib-ldoor
$(ROOT)/usr/lib/llib-lefi.ln:=          REALPATH=../../lib/llib-lefi.ln
$(ROOT)/usr/lib/llib-lefi:=             REALPATH=../../lib/llib-lefi
$(ROOT)/usr/lib/llib-lelf.ln:=          REALPATH=../../lib/llib-lelf.ln
$(ROOT)/usr/lib/llib-lelf:=             REALPATH=../../lib/llib-lelf
$(ROOT)/usr/lib/llib-lfdisk.ln:=        REALPATH=../../lib/llib-lfdisk.ln
$(ROOT)/usr/lib/llib-lfdisk:=           REALPATH=../../lib/llib-lfdisk
$(ROOT)/usr/lib/llib-lgen.ln:=          REALPATH=../../lib/llib-lgen.ln
$(ROOT)/usr/lib/llib-lgen:=             REALPATH=../../lib/llib-lgen
$(ROOT)/usr/lib/llib-linetutil.ln:=     REALPATH=../../lib/llib-linetutil.ln
$(ROOT)/usr/lib/llib-linetutil:=        REALPATH=../../lib/llib-linetutil
$(ROOT)/usr/lib/llib-lintl.ln:=         REALPATH=../../lib/llib-lintl.ln
$(ROOT)/usr/lib/llib-lintl:=            REALPATH=../../lib/llib-lintl
$(ROOT)/usr/lib/llib-lkstat.ln:=        REALPATH=../../lib/llib-lkstat.ln
$(ROOT)/usr/lib/llib-lkstat:=           REALPATH=../../lib/llib-lkstat
$(ROOT)/usr/lib/llib-lm:=               REALPATH=../../lib/llib-lm
$(ROOT)/usr/lib/llib-lm.ln:=            REALPATH=../../lib/llib-lm.ln
$(ROOT)/usr/lib/llib-lmd5.ln:=          REALPATH=../../lib/llib-lmd5.ln
$(ROOT)/usr/lib/llib-lmd5:=             REALPATH=../../lib/llib-lmd5
$(ROOT)/usr/lib/llib-lmeta.ln:=         REALPATH=../../lib/llib-lmeta.ln
$(ROOT)/usr/lib/llib-lmeta:=            REALPATH=../../lib/llib-lmeta
$(ROOT)/usr/lib/llib-lnsl.ln:=          REALPATH=../../lib/llib-lnsl.ln
$(ROOT)/usr/lib/llib-lnsl:=             REALPATH=../../lib/llib-lnsl
$(ROOT)/usr/lib/llib-lnvpair.ln:=       REALPATH=../../lib/llib-lnvpair.ln
$(ROOT)/usr/lib/llib-lnvpair:=          REALPATH=../../lib/llib-lnvpair
$(ROOT)/usr/lib/llib-lpam.ln:=          REALPATH=../../lib/llib-lpam.ln
$(ROOT)/usr/lib/llib-lpam:=             REALPATH=../../lib/llib-lpam
$(ROOT)/usr/lib/llib-lposix4.ln:=       REALPATH=../../lib/llib-lrt.ln
$(ROOT)/usr/lib/llib-lposix4:=          REALPATH=../../lib/llib-lrt
$(ROOT)/usr/lib/llib-lpthread.ln:=      REALPATH=../../lib/llib-lpthread.ln
$(ROOT)/usr/lib/llib-lpthread:=         REALPATH=../../lib/llib-lpthread
$(ROOT)/usr/lib/llib-lresolv.ln:=       REALPATH=../../lib/llib-lresolv.ln
$(ROOT)/usr/lib/llib-lresolv:=          REALPATH=../../lib/llib-lresolv
$(ROOT)/usr/lib/llib-lrpcsvc.ln:=       REALPATH=../../lib/llib-lrpcsvc.ln
$(ROOT)/usr/lib/llib-lrpcsvc:=          REALPATH=../../lib/llib-lrpcsvc
$(ROOT)/usr/lib/llib-lrt.ln:=           REALPATH=../../lib/llib-lrt.ln
$(ROOT)/usr/lib/llib-lrt:=              REALPATH=../../lib/llib-lrt
$(ROOT)/usr/lib/llib-lrtld_db.ln:=      REALPATH=../../lib/llib-lrtld_db.ln
$(ROOT)/usr/lib/llib-lrtld_db:=         REALPATH=../../lib/llib-lrtld_db
$(ROOT)/usr/lib/llib-lscf.ln:=          REALPATH=../../lib/llib-lscf.ln
$(ROOT)/usr/lib/llib-lscf:=             REALPATH=../../lib/llib-lscf
$(ROOT)/usr/lib/llib-lsec.ln:=          REALPATH=../../lib/llib-lsec.ln
$(ROOT)/usr/lib/llib-lsec:=             REALPATH=../../lib/llib-lsec
$(ROOT)/usr/lib/llib-lsecdb.ln:=        REALPATH=../../lib/llib-lsecdb.ln
$(ROOT)/usr/lib/llib-lsecdb:=           REALPATH=../../lib/llib-lsecdb
```



```
$(ROOT)/usr/lib/llib-lsendfile.ln:=     REALPATH=../../lib/llib-lsendfile.ln
$(ROOT)/usr/lib/llib-lsendfile:=        REALPATH=../../lib/llib-lsendfile
$(ROOT)/usr/lib/llib-lsocket.ln:=       REALPATH=../../lib/llib-lsocket.ln
$(ROOT)/usr/lib/llib-lsocket:=          REALPATH=../../lib/llib-lsocket
$(ROOT)/usr/lib/llib-lsysevent.ln:=     REALPATH=../../lib/llib-lsysevent.ln
$(ROOT)/usr/lib/llib-lsysevent:=        REALPATH=../../lib/llib-lsysevent
$(ROOT)/usr/lib/llib-ltermcap.ln:=      REALPATH=../../lib/llib-ltermcap.ln
$(ROOT)/usr/lib/llib-ltermcap:=         REALPATH=../../lib/llib-ltermcap
$(ROOT)/usr/lib/llib-ltermlib.ln:=      REALPATH=../../lib/llib-lcurses.ln
$(ROOT)/usr/lib/llib-ltermlib:=         REALPATH=../../lib/llib-lcurses
$(ROOT)/usr/lib/llib-lthread.ln:=       REALPATH=../../lib/llib-lthread.ln
$(ROOT)/usr/lib/llib-lthread:=          REALPATH=../../lib/llib-lthread
$(ROOT)/usr/lib/llib-lthread_db.ln:=    REALPATH=../../lib/llib-lc_db.ln
$(ROOT)/usr/lib/llib-lthread_db:=       REALPATH=../../lib/llib-lc_db
$(ROOT)/usr/lib/llib-ltsnet.ln:=        REALPATH=../../lib/llib-ltsnet.ln
$(ROOT)/usr/lib/llib-ltsnet:=           REALPATH=../../lib/llib-ltsnet
$(ROOT)/usr/lib/llib-ltsol.ln:=         REALPATH=../../lib/llib-ltsol.ln
$(ROOT)/usr/lib/llib-ltsol:=            REALPATH=../../lib/llib-ltsol
$(ROOT)/usr/lib/llib-lumem.ln:=         REALPATH=../../lib/llib-lumem.ln
$(ROOT)/usr/lib/llib-lumem:=            REALPATH=../../lib/llib-lumem
$(ROOT)/usr/lib/llib-luuid.ln:=         REALPATH=../../lib/llib-luuid.ln
$(ROOT)/usr/lib/llib-luuid:=            REALPATH=../../lib/llib-luuid
$(ROOT)/usr/lib/llib-lxnet.ln:=         REALPATH=../../lib/llib-lxnet.ln
$(ROOT)/usr/lib/llib-lxnet:=            REALPATH=../../lib/llib-lxnet
$(ROOT)/usr/lib/llib-lzfs.ln:=          REALPATH=../../lib/llib-lzfs.ln
$(ROOT)/usr/lib/llib-lzfs:=             REALPATH=../../lib/llib-lzfs
$(ROOT)/usr/lib/llib-lzfs_core.ln:=     REALPATH=../../lib/llib-lzfs_core.ln
$(ROOT)/usr/lib/llib-lzfs_core:=        REALPATH=../../lib/llib-lzfs_core
$(ROOT)/usr/lib/nss_compat.so.1:=       REALPATH=../../lib/nss_compat.so.1
$(ROOT)/usr/lib/nss_dns.so.1:=          REALPATH=../../lib/nss_dns.so.1
$(ROOT)/usr/lib/nss_files.so.1:=        REALPATH=../../lib/nss_files.so.1
$(ROOT)/usr/lib/nss_nis.so.1:=          REALPATH=../../lib/nss_nis.so.1
$(ROOT)/usr/lib/nss_user.so.1:=         REALPATH=../../lib/nss_user.so.1
$(ROOT)/usr/lib/fm/libfmevent.so.1:=    REALPATH=../../../lib/fm/libfmevent.so.1
$(ROOT)/usr/lib/fm/libfmevent.so:=      REALPATH=../../../lib/fm/libfmevent.so.1
$(ROOT)/usr/lib/fm/llib-lfmevent.ln:=   REALPATH=../../../lib/fm/llib-lfmevent.l
$(ROOT)/usr/lib/fm/llib-lfmevent:=      REALPATH=../../../lib/fm/llib-lfmevent

$(ROOT)/lib/$(MACH64)/libposix4.so.1:= \
        REALPATH=librt.so.1
$(ROOT)/lib/$(MACH64)/libposix4.so:= \
        REALPATH=libposix4.so.1
$(ROOT)/lib/$(MACH64)/llib-lposix4.ln:= \
        REALPATH=llib-lrt.ln
$(ROOT)/lib/$(MACH64)/libthread_db.so.1:= \
        REALPATH=libc_db.so.1
$(ROOT)/lib/$(MACH64)/libthread_db.so:= \
        REALPATH=libc_db.so.1
$(ROOT)/usr/lib/$(MACH64)/ld.so.1:= \
        REALPATH=../../../lib/$(MACH64)/ld.so.1
$(ROOT)/usr/lib/$(MACH64)/libadm.so.1:= \
        REALPATH=../../../lib/$(MACH64)/libadm.so.1
$(ROOT)/usr/lib/$(MACH64)/libadm.so:= \
        REALPATH=../../../lib/$(MACH64)/libadm.so.1
$(ROOT)/usr/lib/$(MACH64)/libaio.so.1:= \
        REALPATH=../../../lib/$(MACH64)/libaio.so.1
$(ROOT)/usr/lib/$(MACH64)/libaio.so:= \
        REALPATH=../../../lib/$(MACH64)/libaio.so.1
$(ROOT)/usr/lib/$(MACH64)/libavl.so.1:= \
        REALPATH=../../../lib/$(MACH64)/libavl.so.1
$(ROOT)/usr/lib/$(MACH64)/libavl.so:= \
        REALPATH=../../../lib/$(MACH64)/libavl.so.1
$(ROOT)/usr/lib/$(MACH64)/libbsm.so.1:= \
        REALPATH=../../../lib/$(MACH64)/libbsm.so.1
$(ROOT)/usr/lib/$(MACH64)/libbsm.so:= \
        REALPATH=../../../lib/$(MACH64)/libbsm.so.1
```

```
$(ROOT)/usr/lib/$(MACH64)/libc.so.1:= \
        REALPATH=../../../lib/$(MACH64)/libc.so.1
$(ROOT)/usr/lib/$(MACH64)/libc.so:= \
        REALPATH=../../../lib/$(MACH64)/libc.so.1
$(ROOT)/usr/lib/$(MACH64)/libc_db.so.1:= \
        REALPATH=../../../lib/$(MACH64)/libc_db.so.1
$(ROOT)/usr/lib/$(MACH64)/libc_db.so:= \
        REALPATH=../../../lib/$(MACH64)/libc_db.so.1
$(ROOT)/usr/lib/$(MACH64)/libcmdutils.so.1:= \
        REALPATH=../../../lib/$(MACH64)/libcmdutils.so.1
$(ROOT)/usr/lib/$(MACH64)/libcmdutils.so:= \
        REALPATH=../../../lib/$(MACH64)/libcmdutils.so.1
$(ROOT)/usr/lib/$(MACH64)/libcontract.so.1:= \
        REALPATH=../../../lib/$(MACH64)/libcontract.so.1
$(ROOT)/usr/lib/$(MACH64)/libcontract.so:= \
        REALPATH=../../../lib/$(MACH64)/libcontract.so.1
$(ROOT)/usr/lib/$(MACH64)/libctf.so.1:= \
        REALPATH=../../../lib/$(MACH64)/libctf.so.1
$(ROOT)/usr/lib/$(MACH64)/libctf.so:= \
        REALPATH=../../../lib/$(MACH64)/libctf.so.1
$(ROOT)/usr/lib/$(MACH64)/libcurses.so.1:= \
        REALPATH=../../../lib/$(MACH64)/libcurses.so.1
$(ROOT)/usr/lib/$(MACH64)/libcurses.so:= \
        REALPATH=../../../lib/$(MACH64)/libcurses.so.1
$(ROOT)/usr/lib/$(MACH64)/libdevice.so.1:= \
        REALPATH=../../../lib/$(MACH64)/libdevice.so.1
$(ROOT)/usr/lib/$(MACH64)/libdevice.so:= \
        REALPATH=../../../lib/$(MACH64)/libdevice.so.1
$(ROOT)/usr/lib/$(MACH64)/libdevid.so.1:= \
        REALPATH=../../../lib/$(MACH64)/libdevid.so.1
$(ROOT)/usr/lib/$(MACH64)/libdevid.so:= \
        REALPATH=../../../lib/$(MACH64)/libdevid.so.1
$(ROOT)/usr/lib/$(MACH64)/libdevinfo.so.1:= \
        REALPATH=../../../lib/$(MACH64)/libdevinfo.so.1
$(ROOT)/usr/lib/$(MACH64)/libdevinfo.so:= \
        REALPATH=../../../lib/$(MACH64)/libdevinfo.so.1
$(ROOT)/usr/lib/$(MACH64)/libdhcputil.so.1:= \
        REALPATH=../../../lib/$(MACH64)/libdhcputil.so.1
$(ROOT)/usr/lib/$(MACH64)/libdhcputil.so:= \
        REALPATH=../../../lib/$(MACH64)/libdhcputil.so.1
$(ROOT)/usr/lib/$(MACH64)/libdl.so.1:= \
        REALPATH=../../../lib/$(MACH64)/libdl.so.1
$(ROOT)/usr/lib/$(MACH64)/libdl.so:= \
        REALPATH=../../../lib/$(MACH64)/libdl.so.1
$(ROOT)/usr/lib/$(MACH64)/libdlpi.so.1:= \
        REALPATH=../../../lib/$(MACH64)/libdlpi.so.1
$(ROOT)/usr/lib/$(MACH64)/libdlpi.so:= \
        REALPATH=../../../lib/$(MACH64)/libdlpi.so.1
$(ROOT)/usr/lib/$(MACH64)/libdoor.so.1:= \
        REALPATH=../../../lib/$(MACH64)/libdoor.so.1
$(ROOT)/usr/lib/$(MACH64)/libdoor.so:= \
        REALPATH=../../../lib/$(MACH64)/libdoor.so.1
$(ROOT)/usr/lib/$(MACH64)/libefi.so.1:= \
        REALPATH=../../../lib/$(MACH64)/libefi.so.1
$(ROOT)/usr/lib/$(MACH64)/libefi.so:= \
        REALPATH=../../../lib/$(MACH64)/libefi.so.1
$(ROOT)/usr/lib/$(MACH64)/libelf.so.1:= \
        REALPATH=../../../lib/$(MACH64)/libelf.so.1
$(ROOT)/usr/lib/$(MACH64)/libelf.so:= \
        REALPATH=../../../lib/$(MACH64)/libelf.so.1
$(ROOT)/usr/lib/$(MACH64)/libgen.so.1:= \
        REALPATH=../../../lib/$(MACH64)/libgen.so.1
$(ROOT)/usr/lib/$(MACH64)/libgen.so:= \
        REALPATH=../../../lib/$(MACH64)/libgen.so.1
$(ROOT)/usr/lib/$(MACH64)/libinetutil.so.1:= \
        REALPATH=../../../lib/$(MACH64)/libinetutil.so.1
```



```
$(ROOT)/usr/lib/$(MACH64)/libinetutil.so:= \
        REALPATH=../../../lib/$(MACH64)/libinetutil.so.1
$(ROOT)/usr/lib/$(MACH64)/libintl.so.1:= \
        REALPATH=../../../lib/$(MACH64)/libintl.so.1
$(ROOT)/usr/lib/$(MACH64)/libintl.so:= \
        REALPATH=../../../lib/$(MACH64)/libintl.so.1
$(ROOT)/usr/lib/$(MACH64)/libkstat.so.1:= \
        REALPATH=../../../lib/$(MACH64)/libkstat.so.1
$(ROOT)/usr/lib/$(MACH64)/libkstat.so:= \
        REALPATH=../../../lib/$(MACH64)/libkstat.so.1
$(ROOT)/usr/lib/$(MACH64)/liblddbg.so.4:= \
        REALPATH=../../../lib/$(MACH64)/liblddbg.so.4
$(ROOT)/usr/lib/$(MACH64)/libm.so.1:= \
        REALPATH=../../../lib/$(MACH64)/libm.so.1
$(ROOT)/usr/lib/$(MACH64)/libm.so.2:= \
        REALPATH=../../../lib/$(MACH64)/libm.so.2
$(ROOT)/usr/lib/$(MACH64)/libm.so:= \
        REALPATH=../../../lib/$(MACH64)/libm.so.2
$(ROOT)/usr/lib/$(MACH64)/libmd.so.1:= \
        REALPATH=../../../lib/$(MACH64)/libmd.so.1
$(ROOT)/usr/lib/$(MACH64)/libmd.so:= \
        REALPATH=../../../lib/$(MACH64)/libmd.so.1
$(ROOT)/usr/lib/$(MACH64)/libmd5.so.1:= \
        REALPATH=../../../lib/$(MACH64)/libmd5.so.1
$(ROOT)/usr/lib/$(MACH64)/libmd5.so:= \
        REALPATH=../../../lib/$(MACH64)/libmd5.so.1
$(ROOT)/usr/lib/$(MACH64)/libmp.so.2:= \
        REALPATH=../../../lib/$(MACH64)/libmp.so.2
$(ROOT)/usr/lib/$(MACH64)/libmp.so:= \
        REALPATH=../../../lib/$(MACH64)/libmp.so.2
$(ROOT)/usr/lib/$(MACH64)/libmvec.so.1:= \
        REALPATH=../../../lib/$(MACH64)/libmvec.so.1
$(ROOT)/usr/lib/$(MACH64)/libmvec.so:= \
        REALPATH=../../../lib/$(MACH64)/libmvec.so.1
$(ROOT)/usr/lib/$(MACH64)/libnsl.so.1:= \
        REALPATH=../../../lib/$(MACH64)/libnsl.so.1
$(ROOT)/usr/lib/$(MACH64)/libnsl.so:= \
        REALPATH=../../../lib/$(MACH64)/libnsl.so.1
$(ROOT)/usr/lib/$(MACH64)/libnvpair.so.1:= \
        REALPATH=../../../lib/$(MACH64)/libnvpair.so.1
$(ROOT)/usr/lib/$(MACH64)/libnvpair.so:= \
        REALPATH=../../../lib/$(MACH64)/libnvpair.so.1
$(ROOT)/usr/lib/$(MACH64)/libpam.so.1:= \
        REALPATH=../../../lib/$(MACH64)/libpam.so.1
$(ROOT)/usr/lib/$(MACH64)/libpam.so:= \
        REALPATH=../../../lib/$(MACH64)/libpam.so.1
$(ROOT)/usr/lib/$(MACH64)/libposix4.so.1:= \
        REALPATH=../../../lib/$(MACH64)/librt.so.1
$(ROOT)/usr/lib/$(MACH64)/libposix4.so:= \
        REALPATH=../../../lib/$(MACH64)/librt.so.1
$(ROOT)/usr/lib/$(MACH64)/libproc.so.1:= \
        REALPATH=../../../lib/$(MACH64)/libproc.so.1
$(ROOT)/usr/lib/$(MACH64)/libproc.so:= \
        REALPATH=../../../lib/$(MACH64)/libproc.so.1
$(ROOT)/usr/lib/$(MACH64)/libpthread.so.1:= \
        REALPATH=../../../lib/$(MACH64)/libpthread.so.1
$(ROOT)/usr/lib/$(MACH64)/libpthread.so:= \
        REALPATH=../../../lib/$(MACH64)/libpthread.so.1
$(ROOT)/usr/lib/$(MACH64)/librcm.so.1:= \
        REALPATH=../../../lib/$(MACH64)/librcm.so.1
$(ROOT)/usr/lib/$(MACH64)/librcm.so:= \
        REALPATH=../../../lib/$(MACH64)/librcm.so.1
$(ROOT)/usr/lib/$(MACH64)/libresolv.so.2:= \
        REALPATH=../../../lib/$(MACH64)/libresolv.so.2
$(ROOT)/usr/lib/$(MACH64)/libresolv.so:= \
        REALPATH=../../../lib/$(MACH64)/libresolv.so.2
```

```
$(ROOT)/usr/lib/$(MACH64)/librestart.so.1:= \
        REALPATH=../../../lib/$(MACH64)/librestart.so.1
$(ROOT)/usr/lib/$(MACH64)/librestart.so:= \
        REALPATH=../../../lib/$(MACH64)/librestart.so.1
$(ROOT)/usr/lib/$(MACH64)/librpcsvc.so.1:= \
        REALPATH=../../../lib/$(MACH64)/librpcsvc.so.1
$(ROOT)/usr/lib/$(MACH64)/librpcsvc.so:= \
        REALPATH=../../../lib/$(MACH64)/librpcsvc.so.1
$(ROOT)/usr/lib/$(MACH64)/librt.so.1:= \
        REALPATH=../../../lib/$(MACH64)/librt.so.1
$(ROOT)/usr/lib/$(MACH64)/librt.so:= \
        REALPATH=../../../lib/$(MACH64)/librt.so.1
$(ROOT)/usr/lib/$(MACH64)/librtld.so.1:= \
        REALPATH=../../../lib/$(MACH64)/librtld.so.1
$(ROOT)/usr/lib/$(MACH64)/librtld_db.so.1:= \
        REALPATH=../../../lib/$(MACH64)/librtld_db.so.1
$(ROOT)/usr/lib/$(MACH64)/librtld_db.so:= \
        REALPATH=../../../lib/$(MACH64)/librtld_db.so.1
$(ROOT)/usr/lib/$(MACH64)/libscf.so.1:= \
        REALPATH=../../../lib/$(MACH64)/libscf.so.1
$(ROOT)/usr/lib/$(MACH64)/libscf.so:= \
        REALPATH=../../../lib/$(MACH64)/libscf.so.1
$(ROOT)/usr/lib/$(MACH64)/libsec.so.1:= \
        REALPATH=../../../lib/$(MACH64)/libsec.so.1
$(ROOT)/usr/lib/$(MACH64)/libsec.so:= \
        REALPATH=../../../lib/$(MACH64)/libsec.so.1
$(ROOT)/usr/lib/$(MACH64)/libsecdb.so.1:= \
        REALPATH=../../../lib/$(MACH64)/libsecdb.so.1
$(ROOT)/usr/lib/$(MACH64)/libsecdb.so:= \
        REALPATH=../../../lib/$(MACH64)/libsecdb.so.1
$(ROOT)/usr/lib/$(MACH64)/libsendfile.so.1:= \
        REALPATH=../../../lib/$(MACH64)/libsendfile.so.1
$(ROOT)/usr/lib/$(MACH64)/libsendfile.so:= \
        REALPATH=../../../lib/$(MACH64)/libsendfile.so.1
$(ROOT)/usr/lib/$(MACH64)/libsocket.so.1:= \
        REALPATH=../../../lib/$(MACH64)/libsocket.so.1
$(ROOT)/usr/lib/$(MACH64)/libsocket.so:= \
        REALPATH=../../../lib/$(MACH64)/libsocket.so.1
$(ROOT)/usr/lib/$(MACH64)/libsysevent.so.1:= \
        REALPATH=../../../lib/$(MACH64)/libsysevent.so.1
$(ROOT)/usr/lib/$(MACH64)/libsysevent.so:= \
        REALPATH=../../../lib/$(MACH64)/libsysevent.so.1
$(ROOT)/usr/lib/$(MACH64)/libtermcap.so.1:= \
        REALPATH=../../../lib/$(MACH64)/libtermcap.so.1
$(ROOT)/usr/lib/$(MACH64)/libtermcap.so:= \
        REALPATH=../../../lib/$(MACH64)/libtermcap.so.1
$(ROOT)/usr/lib/$(MACH64)/libtermlib.so.1:= \
        REALPATH=../../../lib/$(MACH64)/libcurses.so.1
$(ROOT)/usr/lib/$(MACH64)/libtermlib.so:= \
        REALPATH=../../../lib/$(MACH64)/libcurses.so.1
$(ROOT)/usr/lib/$(MACH64)/libthread.so.1:= \
        REALPATH=../../../lib/$(MACH64)/libthread.so.1
$(ROOT)/usr/lib/$(MACH64)/libthread.so:= \
        REALPATH=../../../lib/$(MACH64)/libthread.so.1
$(ROOT)/usr/lib/$(MACH64)/libthread_db.so.1:= \
        REALPATH=../../../lib/$(MACH64)/libc_db.so.1
$(ROOT)/usr/lib/$(MACH64)/libthread_db.so:= \
        REALPATH=../../../lib/$(MACH64)/libc_db.so.1
$(ROOT)/usr/lib/$(MACH64)/libtsnet.so.1:= \
        REALPATH=../../../lib/$(MACH64)/libtsnet.so.1
$(ROOT)/usr/lib/$(MACH64)/libtsnet.so:= \
        REALPATH=../../../lib/$(MACH64)/libtsnet.so.1
$(ROOT)/usr/lib/$(MACH64)/libtsol.so.2:= \
        REALPATH=../../../lib/$(MACH64)/libtsol.so.2
$(ROOT)/usr/lib/$(MACH64)/libtsol.so:= \
        REALPATH=../../../lib/$(MACH64)/libtsol.so.2
```



```
$(ROOT)/usr/lib/$(MACH64)/libumem.so.1:= \
        REALPATH=../../../lib/$(MACH64)/libumem.so.1
$(ROOT)/usr/lib/$(MACH64)/libumem.so:= \
        REALPATH=../../../lib/$(MACH64)/libumem.so.1
$(ROOT)/usr/lib/$(MACH64)/libuuid.so.1:= \
        REALPATH=../../../lib/$(MACH64)/libuuid.so.1
$(ROOT)/usr/lib/$(MACH64)/libuuid.so:= \
        REALPATH=../../../lib/$(MACH64)/libuuid.so.1
$(ROOT)/usr/lib/$(MACH64)/libuutil.so.1:= \
        REALPATH=../../../lib/$(MACH64)/libuutil.so.1
$(ROOT)/usr/lib/$(MACH64)/libuutil.so:= \
        REALPATH=../../../lib/$(MACH64)/libuutil.so.1
$(ROOT)/usr/lib/$(MACH64)/libw.so.1:= \
        REALPATH=../../../lib/$(MACH64)/libw.so.1
$(ROOT)/usr/lib/$(MACH64)/libw.so:= \
        REALPATH=../../../lib/$(MACH64)/libw.so.1
$(ROOT)/usr/lib/$(MACH64)/libxnet.so.1:= \
        REALPATH=../../../lib/$(MACH64)/libxnet.so.1
$(ROOT)/usr/lib/$(MACH64)/libxnet.so:= \
        REALPATH=../../../lib/$(MACH64)/libxnet.so.1
$(ROOT)/usr/lib/$(MACH64)/libzfs.so:= \
        REALPATH=../../../lib/$(MACH64)/libzfs.so.1
$(ROOT)/usr/lib/$(MACH64)/libzfs.so.1:= \
        REALPATH=../../../lib/$(MACH64)/libzfs.so.1
$(ROOT)/usr/lib/$(MACH64)/libzfs_core.so:= \
        REALPATH=../../../lib/$(MACH64)/libzfs_core.so.1
$(ROOT)/usr/lib/$(MACH64)/libzfs_core.so.1:= \
        REALPATH=../../../lib/$(MACH64)/libzfs_core.so.1
$(ROOT)/usr/lib/$(MACH64)/libfdisk.so.1:= \
        REALPATH=../../../lib/$(MACH64)/libfdisk.so.1
$(ROOT)/usr/lib/$(MACH64)/libfdisk.so:= \
        REALPATH=../../../lib/$(MACH64)/libfdisk.so.1
$(ROOT)/usr/lib/$(MACH64)/llib-ladm.ln:= \
        REALPATH=../../../lib/$(MACH64)/llib-ladm.ln
$(ROOT)/usr/lib/$(MACH64)/llib-laio.ln:= \
        REALPATH=../../../lib/$(MACH64)/llib-laio.ln
$(ROOT)/usr/lib/$(MACH64)/llib-lavl.ln:= \
        REALPATH=../../../lib/$(MACH64)/llib-lavl.ln
$(ROOT)/usr/lib/$(MACH64)/llib-lbsm.ln:= \
        REALPATH=../../../lib/$(MACH64)/llib-lbsm.ln
$(ROOT)/usr/lib/$(MACH64)/llib-lc.ln:= \
        REALPATH=../../../lib/$(MACH64)/llib-lc.ln
$(ROOT)/usr/lib/$(MACH64)/llib-lcmdutils.ln:= \
        REALPATH=../../../lib/$(MACH64)/llib-lcmdutils.ln
$(ROOT)/usr/lib/$(MACH64)/llib-lcontract.ln:= \
        REALPATH=../../../lib/$(MACH64)/llib-lcontract.ln
$(ROOT)/usr/lib/$(MACH64)/llib-lctf.ln:= \
        REALPATH=../../../lib/$(MACH64)/llib-lctf.ln
$(ROOT)/usr/lib/$(MACH64)/llib-lcurses.ln:= \
        REALPATH=../../../lib/$(MACH64)/llib-lcurses.ln
$(ROOT)/usr/lib/$(MACH64)/llib-ldevice.ln:= \
        REALPATH=../../../lib/$(MACH64)/llib-ldevice.ln
$(ROOT)/usr/lib/$(MACH64)/llib-ldevid.ln:= \
        REALPATH=../../../lib/$(MACH64)/llib-ldevid.ln
$(ROOT)/usr/lib/$(MACH64)/llib-ldevinfo.ln:= \
        REALPATH=../../../lib/$(MACH64)/llib-ldevinfo.ln
$(ROOT)/usr/lib/$(MACH64)/llib-ldhcputil.ln:= \
        REALPATH=../../../lib/$(MACH64)/llib-ldhcputil.ln
$(ROOT)/usr/lib/$(MACH64)/llib-ldl.ln:= \
        REALPATH=../../../lib/$(MACH64)/llib-ldl.ln
$(ROOT)/usr/lib/$(MACH64)/llib-ldoor.ln:= \
        REALPATH=../../../lib/$(MACH64)/llib-ldoor.ln
$(ROOT)/usr/lib/$(MACH64)/llib-lefi.ln:= \
        REALPATH=../../../lib/$(MACH64)/llib-lefi.ln
$(ROOT)/usr/lib/$(MACH64)/llib-lelf.ln:= \
        REALPATH=../../../lib/$(MACH64)/llib-lelf.ln
```

```
$(ROOT)/usr/lib/$(MACH64)/llib-lgen.ln:= \
        REALPATH=../../../lib/$(MACH64)/llib-lgen.ln
$(ROOT)/usr/lib/$(MACH64)/llib-linetutil.ln:= \
        REALPATH=../../../lib/$(MACH64)/llib-linetutil.ln
$(ROOT)/usr/lib/$(MACH64)/llib-lintl.ln:= \
        REALPATH=../../../lib/$(MACH64)/llib-lintl.ln
$(ROOT)/usr/lib/$(MACH64)/llib-lkstat.ln:= \
        REALPATH=../../../lib/$(MACH64)/llib-lkstat.ln
$(ROOT)/usr/lib/$(MACH64)/llib-lm.ln:= \
        REALPATH=../../../lib/$(MACH64)/llib-lm.ln
$(ROOT)/usr/lib/$(MACH64)/llib-lmd5.ln:= \
        REALPATH=../../../lib/$(MACH64)/llib-lmd5.ln
$(ROOT)/usr/lib/$(MACH64)/llib-lnsl.ln:= \
        REALPATH=../../../lib/$(MACH64)/llib-lnsl.ln
$(ROOT)/usr/lib/$(MACH64)/llib-lnvpair.ln:= \
        REALPATH=../../../lib/$(MACH64)/llib-lnvpair.ln
$(ROOT)/usr/lib/$(MACH64)/llib-lpam.ln:= \
        REALPATH=../../../lib/$(MACH64)/llib-lpam.ln
$(ROOT)/usr/lib/$(MACH64)/llib-lposix4.ln:= \
        REALPATH=../../../lib/$(MACH64)/llib-lrt.ln
$(ROOT)/usr/lib/$(MACH64)/llib-lpthread.ln:= \
        REALPATH=../../../lib/$(MACH64)/llib-lpthread.ln
$(ROOT)/usr/lib/$(MACH64)/llib-lresolv.ln:= \
        REALPATH=../../../lib/$(MACH64)/llib-lresolv.ln
$(ROOT)/usr/lib/$(MACH64)/llib-lrpcsvc.ln:= \
        REALPATH=../../../lib/$(MACH64)/llib-lrpcsvc.ln
$(ROOT)/usr/lib/$(MACH64)/llib-lrt.ln:= \
        REALPATH=../../../lib/$(MACH64)/llib-lrt.ln
$(ROOT)/usr/lib/$(MACH64)/llib-lrtld_db.ln:= \
        REALPATH=../../../lib/$(MACH64)/llib-lrtld_db.ln
$(ROOT)/usr/lib/$(MACH64)/llib-lscf.ln:= \
        REALPATH=../../../lib/$(MACH64)/llib-lscf.ln
$(ROOT)/usr/lib/$(MACH64)/llib-lsec.ln:= \
        REALPATH=../../../lib/$(MACH64)/llib-lsec.ln
$(ROOT)/usr/lib/$(MACH64)/llib-lsecdb.ln:= \
        REALPATH=../../../lib/$(MACH64)/llib-lsecdb.ln
$(ROOT)/usr/lib/$(MACH64)/llib-lsendfile.ln:= \
        REALPATH=../../../lib/$(MACH64)/llib-lsendfile.ln
$(ROOT)/usr/lib/$(MACH64)/llib-lsocket.ln:= \
        REALPATH=../../../lib/$(MACH64)/llib-lsocket.ln
$(ROOT)/usr/lib/$(MACH64)/llib-lsysevent.ln:= \
        REALPATH=../../../lib/$(MACH64)/llib-lsysevent.ln
$(ROOT)/usr/lib/$(MACH64)/llib-ltermcap.ln:= \
        REALPATH=../../../lib/$(MACH64)/llib-ltermcap.ln
$(ROOT)/usr/lib/$(MACH64)/llib-ltermlib.ln:= \
        REALPATH=../../../lib/$(MACH64)/llib-lcurses.ln
$(ROOT)/usr/lib/$(MACH64)/llib-lthread.ln:= \
        REALPATH=../../../lib/$(MACH64)/llib-lthread.ln
$(ROOT)/usr/lib/$(MACH64)/llib-lthread_db.ln:= \
        REALPATH=../../../lib/$(MACH64)/llib-lc_db.ln
$(ROOT)/usr/lib/$(MACH64)/llib-ltsnet.ln:= \
        REALPATH=../../../lib/$(MACH64)/llib-ltsnet.ln
$(ROOT)/usr/lib/$(MACH64)/llib-ltsol.ln:= \
        REALPATH=../../../lib/$(MACH64)/llib-ltsol.ln
$(ROOT)/usr/lib/$(MACH64)/llib-lumem.ln:= \
        REALPATH=../../../lib/$(MACH64)/llib-lumem.ln
$(ROOT)/usr/lib/$(MACH64)/llib-luuid.ln:= \
        REALPATH=../../../lib/$(MACH64)/llib-luuid.ln
$(ROOT)/usr/lib/$(MACH64)/llib-lxnet.ln:= \
        REALPATH=../../../lib/$(MACH64)/llib-lxnet.ln
$(ROOT)/usr/lib/$(MACH64)/llib-lzfs.ln:= \
        REALPATH=../../../lib/$(MACH64)/llib-lzfs.ln
$(ROOT)/usr/lib/$(MACH64)/llib-lzfs_core.ln:= \
        REALPATH=../../../lib/$(MACH64)/llib-lzfs_core.ln
$(ROOT)/usr/lib/$(MACH64)/llib-lfdisk.ln:= \
        REALPATH=../../../lib/$(MACH64)/llib-lfdisk.ln
```



```
$(ROOT)/usr/lib/$(MACH64)/nss_compat.so.1:= \
        REALPATH=../../../lib/$(MACH64)/nss_compat.so.1
$(ROOT)/usr/lib/$(MACH64)/nss_dns.so.1:= \
        REALPATH=../../../lib/$(MACH64)/nss_dns.so.1
$(ROOT)/usr/lib/$(MACH64)/nss_files.so.1:= \
        REALPATH=../../../lib/$(MACH64)/nss_files.so.1
$(ROOT)/usr/lib/$(MACH64)/nss_nis.so.1:= \
        REALPATH=../../../lib/$(MACH64)/nss_nis.so.1
$(ROOT)/usr/lib/$(MACH64)/nss_user.so.1:= \
        REALPATH=../../../lib/$(MACH64)/nss_user.so.1
$(ROOT)/usr/lib/fm/$(MACH64)/libfmevent.so.1:= \
        REALPATH=../../../../lib/fm/$(MACH64)/libfmevent.so.1
$(ROOT)/usr/lib/fm/$(MACH64)/libfmevent.so:= \
        REALPATH=../../../../lib/fm/$(MACH64)/libfmevent.so.1
$(ROOT)/usr/lib/fm/$(MACH64)/llib-lfmevent.ln:= \
        REALPATH=../../../../lib/fm/$(MACH64)/llib-lfmevent.ln

i386_SYM.USRLIB= \
        /usr/lib/libfdisk.so \
        /usr/lib/libfdisk.so.1 \
        /usr/lib/llib-lfdisk \
        /usr/lib/llib-lfdisk.ln

SYM.USRLIB= \
        $($(MACH)_SYM.USRLIB)   \
        /lib/libposix4.so \
        /lib/libposix4.so.1 \
        /lib/llib-lposix4 \
        /lib/llib-lposix4.ln \
        /lib/libthread_db.so \
        /lib/libthread_db.so.1 \
        /usr/lib/ld.so.1 \
        /usr/lib/libadm.so \
        /usr/lib/libadm.so.1 \
        /usr/lib/libaio.so \
        /usr/lib/libaio.so.1 \
        /usr/lib/libavl.so \
        /usr/lib/libavl.so.1 \
        /usr/lib/libbsm.so \
        /usr/lib/libbsm.so.1 \
        /usr/lib/libc.so \
        /usr/lib/libc.so.1 \
        /usr/lib/libc_db.so \
        /usr/lib/libc_db.so.1 \
        /usr/lib/libcmdutils.so \
        /usr/lib/libcmdutils.so.1 \
        /usr/lib/libcontract.so \
        /usr/lib/libcontract.so.1 \
        /usr/lib/libctf.so \
        /usr/lib/libctf.so.1 \
        /usr/lib/libcurses.so \
        /usr/lib/libcurses.so.1 \
        /usr/lib/libdevice.so \
        /usr/lib/libdevice.so.1 \
        /usr/lib/libdevid.so \
        /usr/lib/libdevid.so.1 \
        /usr/lib/libdevinfo.so \
        /usr/lib/libdevinfo.so.1 \
        /usr/lib/libdhcpagent.so \
        /usr/lib/libdhcpagent.so.1 \
        /usr/lib/libdhcputil.so \
        /usr/lib/libdhcputil.so.1 \
        /usr/lib/libdl.so \
        /usr/lib/libdl.so.1 \
        /usr/lib/libdlpi.so \
        /usr/lib/libdlpi.so.1 \
```

```
1581         /usr/lib/libdoor.so \
1582         /usr/lib/libdoor.so.1 \
1583         /usr/lib/libefi.so \
1584         /usr/lib/libefi.so.1 \
1585         /usr/lib/libelf.so \
1586         /usr/lib/libelf.so.1 \
1587         /usr/lib/libgen.so \
1588         /usr/lib/libgen.so.1 \
1589         /usr/lib/libinetutil.so \
1590         /usr/lib/libinetutil.so.1 \
1591         /usr/lib/libintl.so \
1592         /usr/lib/libintl.so.1 \
1593         /usr/lib/libkstat.so \
1594         /usr/lib/libkstat.so.1 \
1595         /usr/lib/liblddbg.so.4 \
1596         /usr/lib/libm.so.1 \
1597         /usr/lib/libm.so.2 \
1598         /usr/lib/libm.so \
1599         /usr/lib/libmd.so \
1600         /usr/lib/libmd.so.1 \
1601         /usr/lib/libmd5.so \
1602         /usr/lib/libmd5.so.1 \
1603         /usr/lib/libmeta.so \
1604         /usr/lib/libmeta.so.1 \
1605         /usr/lib/libmp.so \
1606         /usr/lib/libmp.so.1 \
1607         /usr/lib/libmp.so.2 \
1608         /usr/lib/libmvec.so.1 \
1609         /usr/lib/libmvec.so \
1610         /usr/lib/libnsl.so \
1611         /usr/lib/libnsl.so.1 \
1612         /usr/lib/libnvpair.so \
1613         /usr/lib/libnvpair.so.1 \
1614         /usr/lib/libpam.so \
1615         /usr/lib/libpam.so.1 \
1616         /usr/lib/libposix4.so \
1617         /usr/lib/libposix4.so.1 \
1618         /usr/lib/libproc.so \
1619         /usr/lib/libproc.so.1 \
1620         /usr/lib/libpthread.so \
1621         /usr/lib/libpthread.so.1 \
1622         /usr/lib/librcm.so \
1623         /usr/lib/librcm.so.1 \
1624         /usr/lib/libresolv.so \
1625         /usr/lib/libresolv.so.1 \
1626         /usr/lib/libresolv.so.2 \
1627         /usr/lib/librestart.so \
1628         /usr/lib/librestart.so.1 \
1629         /usr/lib/librpcsvc.so \
1630         /usr/lib/librpcsvc.so.1 \
1631         /usr/lib/librt.so \
1632         /usr/lib/librt.so.1 \
1633         /usr/lib/librtld.so.1 \
1634         /usr/lib/librtld_db.so \
1635         /usr/lib/librtld_db.so.1 \
1636         /usr/lib/libscf.so \
1637         /usr/lib/libscf.so.1 \
1638         /usr/lib/libsec.so \
1639         /usr/lib/libsec.so.1 \
1640         /usr/lib/libsecdb.so \
1641         /usr/lib/libsecdb.so.1 \
1642         /usr/lib/libsendfile.so \
1643         /usr/lib/libsendfile.so.1 \
1644         /usr/lib/libsocket.so \
1645         /usr/lib/libsocket.so.1 \
1646         /usr/lib/libsysevent.so \
```



```
1647         /usr/lib/libsysevent.so.1 \
1648         /usr/lib/libtermcap.so \
1649         /usr/lib/libtermcap.so.1 \
1650         /usr/lib/libtermlib.so \
1651         /usr/lib/libtermlib.so.1 \
1652         /usr/lib/libthread.so \
1653         /usr/lib/libthread.so.1 \
1654         /usr/lib/libthread_db.so \
1655         /usr/lib/libthread_db.so.1 \
1656         /usr/lib/libtsnet.so \
1657         /usr/lib/libtsnet.so.1 \
1658         /usr/lib/libtsol.so \
1659         /usr/lib/libtsol.so.2 \
1660         /usr/lib/libumem.so \
1661         /usr/lib/libumem.so.1 \
1662         /usr/lib/libuuid.so \
1663         /usr/lib/libuuid.so.1 \
1664         /usr/lib/libuutil.so \
1665         /usr/lib/libuutil.so.1 \
1666         /usr/lib/libw.so \
1667         /usr/lib/libw.so.1 \
1668         /usr/lib/libxnet.so \
1669         /usr/lib/libxnet.so.1 \
1670         /usr/lib/libzfs.so \
1671         /usr/lib/libzfs.so.1 \
1672         /usr/lib/libzfs_core.so \
1673         /usr/lib/libzfs_core.so.1 \
1674         /usr/lib/llib-ladm \
1675         /usr/lib/llib-ladm.ln \
1676         /usr/lib/llib-laio \
1677         /usr/lib/llib-laio.ln \
1678         /usr/lib/llib-lavl \
1679         /usr/lib/llib-lavl.ln \
1680         /usr/lib/llib-lbsm \
1681         /usr/lib/llib-lbsm.ln \
1682         /usr/lib/llib-lc \
1683         /usr/lib/llib-lc.ln \
1684         /usr/lib/llib-lcmdutils \
1685         /usr/lib/llib-lcmdutils.ln \
1686         /usr/lib/llib-lcontract \
1687         /usr/lib/llib-lcontract.ln \
1688         /usr/lib/llib-lctf \
1689         /usr/lib/llib-lctf.ln \
1690         /usr/lib/llib-lcurses \
1691         /usr/lib/llib-lcurses.ln \
1692         /usr/lib/llib-ldevice \
1693         /usr/lib/llib-ldevice.ln \
1694         /usr/lib/llib-ldevid \
1695         /usr/lib/llib-ldevid.ln \
1696         /usr/lib/llib-ldevinfo \
1697         /usr/lib/llib-ldevinfo.ln \
1698         /usr/lib/llib-ldhcpagent \
1699         /usr/lib/llib-ldhcpagent.ln \
1700         /usr/lib/llib-ldhcputil \
1701         /usr/lib/llib-ldhcputil.ln \
1702         /usr/lib/llib-ldl \
1703         /usr/lib/llib-ldl.ln \
1704         /usr/lib/llib-ldoor \
1705         /usr/lib/llib-ldoor.ln \
1706         /usr/lib/llib-lefi \
1707         /usr/lib/llib-lefi.ln \
1708         /usr/lib/llib-lelf \
1709         /usr/lib/llib-lelf.ln \
1710         /usr/lib/llib-lgen \
1711         /usr/lib/llib-lgen.ln \
1712         /usr/lib/llib-linetutil \
```

```
1713         /usr/lib/llib-linetutil.ln \
1714         /usr/lib/llib-lintl \
1715         /usr/lib/llib-lintl.ln \
1716         /usr/lib/llib-lkstat \
1717         /usr/lib/llib-lkstat.ln \
1718         /usr/lib/llib-lm \
1719         /usr/lib/llib-lm.ln \
1720         /usr/lib/llib-lmd5 \
1721         /usr/lib/llib-lmd5.ln \
1722         /usr/lib/llib-lmeta \
1723         /usr/lib/llib-lmeta.ln \
1724         /usr/lib/llib-lnsl \
1725         /usr/lib/llib-lnsl.ln \
1726         /usr/lib/llib-lnvpair \
1727         /usr/lib/llib-lnvpair.ln \
1728         /usr/lib/llib-lpam \
1729         /usr/lib/llib-lpam.ln \
1730         /usr/lib/llib-lposix4 \
1731         /usr/lib/llib-lposix4.ln \
1732         /usr/lib/llib-lpthread \
1733         /usr/lib/llib-lpthread.ln \
1734         /usr/lib/llib-lresolv \
1735         /usr/lib/llib-lresolv.ln \
1736         /usr/lib/llib-lrpcsvc \
1737         /usr/lib/llib-lrpcsvc.ln \
1738         /usr/lib/llib-lrt \
1739         /usr/lib/llib-lrt.ln \
1740         /usr/lib/llib-lrtld_db \
1741         /usr/lib/llib-lrtld_db.ln \
1742         /usr/lib/llib-lscf \
1743         /usr/lib/llib-lscf.ln \
1744         /usr/lib/llib-lsec \
1745         /usr/lib/llib-lsec.ln \
1746         /usr/lib/llib-lsecdb \
1747         /usr/lib/llib-lsecdb.ln \
1748         /usr/lib/llib-lsendfile \
1749         /usr/lib/llib-lsendfile.ln \
1750         /usr/lib/llib-lsocket \
1751         /usr/lib/llib-lsocket.ln \
1752         /usr/lib/llib-lsysevent \
1753         /usr/lib/llib-lsysevent.ln \
1754         /usr/lib/llib-ltermcap \
1755         /usr/lib/llib-ltermcap.ln \
1756         /usr/lib/llib-ltermlib \
1757         /usr/lib/llib-ltermlib.ln \
1758         /usr/lib/llib-lthread \
1759         /usr/lib/llib-lthread.ln \
1760         /usr/lib/llib-lthread_db \
1761         /usr/lib/llib-lthread_db.ln \
1762         /usr/lib/llib-ltsnet \
1763         /usr/lib/llib-ltsnet.ln \
1764         /usr/lib/llib-ltsol \
1765         /usr/lib/llib-ltsol.ln \
1766         /usr/lib/llib-lumem \
1767         /usr/lib/llib-lumem.ln \
1768         /usr/lib/llib-luuid \
1769         /usr/lib/llib-luuid.ln \
1770         /usr/lib/llib-lxnet \
1771         /usr/lib/llib-lxnet.ln \
1772         /usr/lib/llib-lzfs \
1773         /usr/lib/llib-lzfs.ln \
1774         /usr/lib/llib-lzfs_core \
1775         /usr/lib/llib-lzfs_core.ln \
1776         /usr/lib/nss_compat.so.1 \
1777         /usr/lib/nss_dns.so.1 \
1778         /usr/lib/nss_files.so.1 \
```



```
1779         /usr/lib/nss_nis.so.1 \
1780         /usr/lib/nss_user.so.1 \
1781         /usr/lib/fm/libfmevent.so \
1782         /usr/lib/fm/libfmevent.so.1 \
1783         /usr/lib/fm/llib-lfmevent \
1784         /usr/lib/fm/llib-lfmevent.ln

1786 sparcv9_SYM.USRLIB64=

1788 amd64_SYM.USRLIB64=     \
1789         /usr/lib/amd64/libfdisk.so      \
1790         /usr/lib/amd64/libfdisk.so.1    \
1791         /usr/lib/amd64/llib-lfdisk.ln


1794 SYM.USRLIB64= \
1795         $($(MACH64)_SYM.USRLIB64)               \
1796         /lib/$(MACH64)/libposix4.so \
1797         /lib/$(MACH64)/libposix4.so.1 \
1798         /lib/$(MACH64)/llib-lposix4.ln \
1799         /lib/$(MACH64)/libthread_db.so \
1800         /lib/$(MACH64)/libthread_db.so.1 \
1801         /usr/lib/$(MACH64)/ld.so.1 \
1802         /usr/lib/$(MACH64)/libadm.so \
1803         /usr/lib/$(MACH64)/libadm.so.1 \
1804         /usr/lib/$(MACH64)/libaio.so \
1805         /usr/lib/$(MACH64)/libaio.so.1 \
1806         /usr/lib/$(MACH64)/libavl.so \
1807         /usr/lib/$(MACH64)/libavl.so.1 \
1808         /usr/lib/$(MACH64)/libbsm.so \
1809         /usr/lib/$(MACH64)/libbsm.so.1 \
1810         /usr/lib/$(MACH64)/libc.so \
1811         /usr/lib/$(MACH64)/libc.so.1 \
1812         /usr/lib/$(MACH64)/libc_db.so \
1813         /usr/lib/$(MACH64)/libc_db.so.1 \
1814         /usr/lib/$(MACH64)/libcmdutils.so \
1815         /usr/lib/$(MACH64)/libcmdutils.so.1 \
1816         /usr/lib/$(MACH64)/libcontract.so \
1817         /usr/lib/$(MACH64)/libcontract.so.1 \
1818         /usr/lib/$(MACH64)/libctf.so \
1819         /usr/lib/$(MACH64)/libctf.so.1 \
1820         /usr/lib/$(MACH64)/libcurses.so \
1821         /usr/lib/$(MACH64)/libcurses.so.1 \
1822         /usr/lib/$(MACH64)/libdevice.so \
1823         /usr/lib/$(MACH64)/libdevice.so.1 \
1824         /usr/lib/$(MACH64)/libdevid.so \
1825         /usr/lib/$(MACH64)/libdevid.so.1 \
1826         /usr/lib/$(MACH64)/libdevinfo.so \
1827         /usr/lib/$(MACH64)/libdevinfo.so.1 \
1828         /usr/lib/$(MACH64)/libdhcputil.so \
1829         /usr/lib/$(MACH64)/libdhcputil.so.1 \
1830         /usr/lib/$(MACH64)/libdl.so \
1831         /usr/lib/$(MACH64)/libdl.so.1 \
1832         /usr/lib/$(MACH64)/libdlpi.so \
1833         /usr/lib/$(MACH64)/libdlpi.so.1 \
1834         /usr/lib/$(MACH64)/libdoor.so \
1835         /usr/lib/$(MACH64)/libdoor.so.1 \
1836         /usr/lib/$(MACH64)/libefi.so \
1837         /usr/lib/$(MACH64)/libefi.so.1 \
1838         /usr/lib/$(MACH64)/libelf.so \
1839         /usr/lib/$(MACH64)/libelf.so.1 \
1840         /usr/lib/$(MACH64)/libgen.so \
1841         /usr/lib/$(MACH64)/libgen.so.1 \
1842         /usr/lib/$(MACH64)/libinetutil.so \
1843         /usr/lib/$(MACH64)/libinetutil.so.1 \
1844         /usr/lib/$(MACH64)/libintl.so \
```

```
1845         /usr/lib/$(MACH64)/libintl.so.1 \
1846         /usr/lib/$(MACH64)/libkstat.so \
1847         /usr/lib/$(MACH64)/libkstat.so.1 \
1848         /usr/lib/$(MACH64)/liblddbg.so.4 \
1849         /usr/lib/$(MACH64)/libm.so.1 \
1850         /usr/lib/$(MACH64)/libm.so.2 \
1851         /usr/lib/$(MACH64)/libm.so \
1852         /usr/lib/$(MACH64)/libmd.so \
1853         /usr/lib/$(MACH64)/libmd.so.1 \
1854         /usr/lib/$(MACH64)/libmd5.so \
1855         /usr/lib/$(MACH64)/libmd5.so.1 \
1856         /usr/lib/$(MACH64)/libmp.so \
1857         /usr/lib/$(MACH64)/libmp.so.2 \
1858         /usr/lib/$(MACH64)/libmvec.so.1 \
1859         /usr/lib/$(MACH64)/libmvec.so \
1860         /usr/lib/$(MACH64)/libnsl.so \
1861         /usr/lib/$(MACH64)/libnsl.so.1 \
1862         /usr/lib/$(MACH64)/libnvpair.so \
1863         /usr/lib/$(MACH64)/libnvpair.so.1 \
1864         /usr/lib/$(MACH64)/libpam.so \
1865         /usr/lib/$(MACH64)/libpam.so.1 \
1866         /usr/lib/$(MACH64)/libposix4.so \
1867         /usr/lib/$(MACH64)/libposix4.so.1 \
1868         /usr/lib/$(MACH64)/libproc.so \
1869         /usr/lib/$(MACH64)/libproc.so.1 \
1870         /usr/lib/$(MACH64)/libpthread.so \
1871         /usr/lib/$(MACH64)/libpthread.so.1 \
1872         /usr/lib/$(MACH64)/librcm.so \
1873         /usr/lib/$(MACH64)/librcm.so.1 \
1874         /usr/lib/$(MACH64)/libresolv.so \
1875         /usr/lib/$(MACH64)/libresolv.so.2 \
1876         /usr/lib/$(MACH64)/librestart.so \
1877         /usr/lib/$(MACH64)/librestart.so.1 \
1878         /usr/lib/$(MACH64)/librpcsvc.so \
1879         /usr/lib/$(MACH64)/librpcsvc.so.1 \
1880         /usr/lib/$(MACH64)/librt.so \
1881         /usr/lib/$(MACH64)/librt.so.1 \
1882         /usr/lib/$(MACH64)/librtld.so.1 \
1883         /usr/lib/$(MACH64)/librtld_db.so \
1884         /usr/lib/$(MACH64)/librtld_db.so.1 \
1885         /usr/lib/$(MACH64)/libscf.so \
1886         /usr/lib/$(MACH64)/libscf.so.1 \
1887         /usr/lib/$(MACH64)/libsec.so \
1888         /usr/lib/$(MACH64)/libsec.so.1 \
1889         /usr/lib/$(MACH64)/libsecdb.so \
1890         /usr/lib/$(MACH64)/libsecdb.so.1 \
1891         /usr/lib/$(MACH64)/libsendfile.so \
1892         /usr/lib/$(MACH64)/libsendfile.so.1 \
1893         /usr/lib/$(MACH64)/libsocket.so \
1894         /usr/lib/$(MACH64)/libsocket.so.1 \
1895         /usr/lib/$(MACH64)/libsysevent.so \
1896         /usr/lib/$(MACH64)/libsysevent.so.1 \
1897         /usr/lib/$(MACH64)/libtermcap.so \
1898         /usr/lib/$(MACH64)/libtermcap.so.1 \
1899         /usr/lib/$(MACH64)/libtermlib.so \
1900         /usr/lib/$(MACH64)/libtermlib.so.1 \
1901         /usr/lib/$(MACH64)/libthread.so \
1902         /usr/lib/$(MACH64)/libthread.so.1 \
1903         /usr/lib/$(MACH64)/libthread_db.so \
1904         /usr/lib/$(MACH64)/libthread_db.so.1 \
1905         /usr/lib/$(MACH64)/libtsnet.so \
1906         /usr/lib/$(MACH64)/libtsnet.so.1 \
1907         /usr/lib/$(MACH64)/libtsol.so \
1908         /usr/lib/$(MACH64)/libtsol.so.2 \
1909         /usr/lib/$(MACH64)/libumem.so \
1910         /usr/lib/$(MACH64)/libumem.so.1 \
```



```
1911         /usr/lib/$(MACH64)/libuuid.so \
1912         /usr/lib/$(MACH64)/libuuid.so.1 \
1913         /usr/lib/$(MACH64)/libuutil.so \
1914         /usr/lib/$(MACH64)/libuutil.so.1 \
1915         /usr/lib/$(MACH64)/libw.so \
1916         /usr/lib/$(MACH64)/libw.so.1 \
1917         /usr/lib/$(MACH64)/libxnet.so \
1918         /usr/lib/$(MACH64)/libxnet.so.1 \
1919         /usr/lib/$(MACH64)/libzfs.so \
1920         /usr/lib/$(MACH64)/libzfs.so.1 \
1921         /usr/lib/$(MACH64)/libzfs_core.so \
1922         /usr/lib/$(MACH64)/libzfs_core.so.1 \
1923         /usr/lib/$(MACH64)/llib-ladm.ln \
1924         /usr/lib/$(MACH64)/llib-laio.ln \
1925         /usr/lib/$(MACH64)/llib-lavl.ln \
1926         /usr/lib/$(MACH64)/llib-lbsm.ln \
1927         /usr/lib/$(MACH64)/llib-lc.ln \
1928         /usr/lib/$(MACH64)/llib-lcmdutils.ln \
1929         /usr/lib/$(MACH64)/llib-lcontract.ln \
1930         /usr/lib/$(MACH64)/llib-lctf.ln \
1931         /usr/lib/$(MACH64)/llib-lcurses.ln \
1932         /usr/lib/$(MACH64)/llib-ldevice.ln \
1933         /usr/lib/$(MACH64)/llib-ldevid.ln \
1934         /usr/lib/$(MACH64)/llib-ldevinfo.ln \
1935         /usr/lib/$(MACH64)/llib-ldhcputil.ln \
1936         /usr/lib/$(MACH64)/llib-ldl.ln \
1937         /usr/lib/$(MACH64)/llib-ldoor.ln \
1938         /usr/lib/$(MACH64)/llib-lefi.ln \
1939         /usr/lib/$(MACH64)/llib-lelf.ln \
1940         /usr/lib/$(MACH64)/llib-lgen.ln \
1941         /usr/lib/$(MACH64)/llib-linetutil.ln \
1942         /usr/lib/$(MACH64)/llib-lintl.ln \
1943         /usr/lib/$(MACH64)/llib-lkstat.ln \
1944         /usr/lib/$(MACH64)/llib-lm.ln \
1945         /usr/lib/$(MACH64)/llib-lmd5.ln \
1946         /usr/lib/$(MACH64)/llib-lnsl.ln \
1947         /usr/lib/$(MACH64)/llib-lnvpair.ln \
1948         /usr/lib/$(MACH64)/llib-lpam.ln \
1949         /usr/lib/$(MACH64)/llib-lposix4.ln \
1950         /usr/lib/$(MACH64)/llib-lpthread.ln \
1951         /usr/lib/$(MACH64)/llib-lresolv.ln \
1952         /usr/lib/$(MACH64)/llib-lrpcsvc.ln \
1953         /usr/lib/$(MACH64)/llib-lrt.ln \
1954         /usr/lib/$(MACH64)/llib-lrtld_db.ln \
1955         /usr/lib/$(MACH64)/llib-lscf.ln \
1956         /usr/lib/$(MACH64)/llib-lsec.ln \
1957         /usr/lib/$(MACH64)/llib-lsecdb.ln \
1958         /usr/lib/$(MACH64)/llib-lsendfile.ln \
1959         /usr/lib/$(MACH64)/llib-lsocket.ln \
1960         /usr/lib/$(MACH64)/llib-lsysevent.ln \
1961         /usr/lib/$(MACH64)/llib-ltermcap.ln \
1962         /usr/lib/$(MACH64)/llib-ltermlib.ln \
1963         /usr/lib/$(MACH64)/llib-lthread.ln \
1964         /usr/lib/$(MACH64)/llib-lthread_db.ln \
1965         /usr/lib/$(MACH64)/llib-ltsnet.ln \
1966         /usr/lib/$(MACH64)/llib-ltsol.ln \
1967         /usr/lib/$(MACH64)/llib-lumem.ln \
1968         /usr/lib/$(MACH64)/llib-luuid.ln \
1969         /usr/lib/$(MACH64)/llib-lxnet.ln \
1970         /usr/lib/$(MACH64)/llib-lzfs.ln \
1971         /usr/lib/$(MACH64)/llib-lzfs_core.ln \
1972         /usr/lib/$(MACH64)/nss_compat.so.1 \
1973         /usr/lib/$(MACH64)/nss_dns.so.1 \
1974         /usr/lib/$(MACH64)/nss_files.so.1 \
1975         /usr/lib/$(MACH64)/nss_nis.so.1 \
1976         /usr/lib/$(MACH64)/nss_user.so.1 \
```

```
1977         /usr/lib/fm/$(MACH64)/libfmevent.so \
1978         /usr/lib/fm/$(MACH64)/libfmevent.so.1 \
1979         /usr/lib/fm/$(MACH64)/llib-lfmevent.ln

1981 #
1982 # usr/src/Makefile uses INS.dir for any member of ROOTDIRS, the fact
1983 # these are symlinks to files has no bearing on this.
1984 #
1985 $(FILELINKS:%=$(ROOT)%):= \
1986     INS.dir= -$(RM) $@; $(SYMLINK) $(REALPATH) $@
```

```
**********************************************************
   13627 Sun May  4 03:04:43 2014
new/usr/src/lib/Makefile
**********************************************************
```

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

# Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
# Copyright (c) 1989, 2010, Oracle and/or its affiliates. All rights reserved.
# Copyright (c) 2012 by Delphix. All rights reserved.
# Copyright (c) 2012, Joyent, Inc. All rights reserved.
# Copyright (c) 2013 Gary Mills

include ../Makefile.master

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

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

SUBDIRS += \
        libc                    .WAIT   \
        ../cmd/sgs/libelf       .WAIT   \
        c_synonyms                      \
        libmd                           \
        libmd5                          \
        librsm                          \
        libmp                   .WAIT   \
        libnsl                  \
        libsecdb                .WAIT   \
```



```
        librpcsvc                       \
        libsocket               .WAIT   \
        libsctp                 \
        libsip                  \
        libcommputil            \
        libresolv                       \
        libresolv2              .WAIT   \
        libw                    .WAIT   \
        libintl                 .WAIT   \
        ../cmd/sgs/librtld_db   \
        libaio                  \
        libast                  \
        libdll                  \
        libcmd                  \
        libshell                \
        libsum                  \
        librt                   \
        libadm                  \
        libctf                  \
        libdtrace               \
        libdtrace_jni           \
        libcurses               \
        libtermcap              \
        libgen                  \
        libgss                  \
        libpam                  \
        libuuid                 \
        libthread               \
        libpthread      .WAIT   \
        libslp          \
        libbsdmalloc    \
        libdoor         \
        libdevinfo      \
        libdladm        \
        libdlpi         \
        libeti          \
        libcrypt        \
        libdns_sd       \
        libefi          \
        libfstyp        \
        libwanboot      \
        libwanbootutil  \
        libcryptoutil   \
        libinetutil     \
        libipadm        \
        libipd          \
        libipmp         \
        libiscsit       \
        libkmf          \
        libkstat        \
        libkvm          \
        liblm           \
        libmalloc       \
        libmapmalloc    \
        libmtmalloc     \
        libnls          \
        libnwam         \
        libsmbios       \
        libtecla        \
        libumem         \
        libnvpair       .WAIT   \
        libexacct       \
        libsasl         \
        libldap5        \
        libsldap        .WAIT   \
        libbsm          \
```

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

157         libpcidb

163 SUBDIRS += \
164         passwdutil      \
165         pam_modules     \
166         crypt_modules   \
167         libadt_jni      \
168         abi             \
169         auditd_plugins  \
170         libvolmgt       \
171         libdevice       \
172         libdevid        \
173         libdhcpsvc      \
174         libc_db         \
175         libndmp         \
176         libsec          \
177         libtnfprobe     \
178         libtnf          \
179         libtnfctl       \
180         libdhcpagent    \
181         libdhcpdu       \
182         libdhcputil     \
183         libxnet         \
184         libipsecutil    \
185         nsswitch        \
186         print           \
187         libuutil        \
188         libscf          \
189         libinetsvc      \
190         librestart      \
191         libsched        \
192         libelfsign      \
193         pkcs11          .WAIT   \
```



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

```
260         libima          \
261         libsun_ima      \
262         mpapi           \
263         librstp         \
264         libreparse      \
265         libhotplug      \
266         libfruutils     .WAIT   \
267         libfru          \
268         $($(MACH)_SUBDIRS)

270 i386_SUBDIRS=           \
271         libntfs         \
272         libparted       \
273         libfdisk        \
274         libsaveargs

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

289 FM_sparc_DEPLIBS= libpri

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

298 #
299 # Create a special version of $(SUBDIRS) with no .WAIT's, for use with the
300 # clean and clobber targets (for more information, see those targets, below).
301 #
302 NOWAIT_SUBDIRS= $(SUBDIRS:.WAIT=)

304 DCSUBDIRS =             \
305         lvm

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



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

373 sparc_MSGSUBDIRS=       \
374         libprtdiag      \
375         libprtdiag_psr

377 i386_MSGSUBDIRS= libfdisk

379 HDRSUBDIRS=             \
380         auditd_plugins  \
381         libast          \
382         libbrand        \
383         libbsm          \
384         libc            \
385         libcmd          \
386         libcmdutils     \
387         libcommputil    \
388         libcontract     \
389         libcpc          \
390         libctf          \
391         libcurses       \
```

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



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

511 i386_HDRSUBDIRS=        \
512         libparted       \
513         libfdisk        \
514         libsaveargs

516 sparc_HDRSUBDIRS=       \
517         libds           \
518         libdscp         \
519         libpri          \
520         libv12n         \
521         storage

523 all :=          TARGET= all
```

```
524 check :=        TARGET= check
525 clean :=        TARGET= clean
526 clobber :=      TARGET= clobber
527 install :=      TARGET= install
528 install_h :=    TARGET= install_h
529 lint :=         TARGET= lint
530 _dc :=          TARGET= _dc
531 _msg :=         TARGET= _msg

533 .KEEP_STATE:

535 #
536 # For the all and install targets, we clearly must respect library
537 # dependencies so that the libraries link correctly.  However, for
538 # the remaining targets (check, clean, clobber, install_h, lint, _dc
539 # and _msg), libraries do not have any dependencies on one another
540 # and thus respecting dependencies just slows down the build.
541 # As such, for these rules, we use pattern replacement to explicitly
542 # avoid triggering the dependency information.  Note that for clean,
543 # clobber and lint, we must use $(NOWAIT_SUBDIRS) rather than
544 # $(SUBDIRS), to prevent '.WAIT' from expanding to '.WAIT-nodepend'.
545 #

547 all:                    $(SUBDIRS)

549 install:                $(SUBDIRS) .WAIT install_extra

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

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

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

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

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

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



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

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


683 $(INTEL_BUILD)libdiskmgt:libfdisk

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

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

705 FRC:
```

```
**********************************************************
   19733 Sun May  4 03:04:45 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-parentheses
516 CERRWARN        += -_gcc=-Wno-unused-variable

518 #endif /* ! codereview */
519 CPPFLAGS        += -DLIBM_BUILD

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

524 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 \
```

```
657         ../common/R/tanhf.c

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



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

730 SRCS_Q_sparc = \
731         $(SRCS_Q)

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

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

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

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

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

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

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



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

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

```
        ../common/C/expm1.c \
        ../common/C/floor.c \
        ../common/C/fmod.c \
        ../common/C/hypot.c \
        ../common/C/ilogb.c \
        ../common/C/isnan.c \
        ../common/C/log.c \
        ../common/C/log10.c \
        ../common/C/log2.c \
        ../common/C/pow.c \
        ../common/C/remainder.c \
        ../common/C/rint.c \
        ../common/C/scalbn.c \
        ../common/C/sin.c \
        ../common/C/sincos.c \
        ../common/C/tan.c

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

SRCS_sparc_sparc = \
        $(SRCS_i386_i386)

SRCS_sparc_sparcv9 = \
        ../common/C/copysign.c \
        ../common/C/fabs.c \
        ../common/C/nextafter.c

SRCS_i386_amd64 = \
        ../common/C/_TBL_atan.c \
        ../common/C/_TBL_exp2.c \
        ../common/C/_TBL_log.c \
        ../common/C/_TBL_log2.c \
        ../common/C/__tan.c \
        ../common/C/_TBL_tan.c \
        ../common/C/copysign.c \
        ../common/C/exp.c \
        ../common/C/fabs.c \
        ../common/C/ilogb.c \
        ../common/C/isnan.c \
        ../common/C/nextafter.c \
        ../common/C/rint.c \
        ../common/C/scalbn.c \
        ../common/C/acos.c \
        ../common/C/asin.c \
        ../common/C/atan.c \
        ../common/C/atan2.c \
        ../common/C/ceil.c \
        ../common/C/cos.c \
        ../common/C/exp10.c \
        ../common/C/exp2.c \
        ../common/C/expm1.c \
        ../common/C/floor.c \
        ../common/C/hypot.c \
        ../common/C/log.c \
        ../common/C/log10.c \
        ../common/C/log2.c \
        ../common/C/pow.c \
        ../common/C/sin.c \
        ../common/C/sincos.c \
        ../common/C/tan.c

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



```
        ../common/C/__lgamma.c \
        ../common/C/__rem_pio2.c \
        ../common/C/__rem_pio2m.c \
        ../common/C/__sin.c \
        ../common/C/__sincos.c \
        ../common/C/__xpg6.c \
        ../common/C/_lib_version.c \
        ../common/C/_SVID_error.c \
        ../common/C/_TBL_ipio2.c \
        ../common/C/_TBL_sin.c \
        ../common/C/acosh.c \
        ../common/C/asinh.c \
        ../common/C/atan2pi.c \
        ../common/C/atanh.c \
        ../common/C/cbrt.c \
        ../common/C/cosh.c \
        ../common/C/erf.c \
        ../common/C/gamma.c \
        ../common/C/gamma_r.c \
        ../common/C/j0.c \
        ../common/C/j1.c \
        ../common/C/jn.c \
        ../common/C/lgamma.c \
        ../common/C/lgamma_r.c \
        ../common/C/log1p.c \
        ../common/C/logb.c \
        ../common/C/matherr.c \
        ../common/C/scalb.c \
        ../common/C/signgam.c \
        ../common/C/significand.c \
        ../common/C/sincospi.c \
        ../common/C/sinh.c \
        ../common/C/sqrt.c \
        ../common/C/tanh.c

SRCS    = \
        $(SRCS_Q_$(MACH)) \
        $(SRCS_LD_$(MACH)) \
        $(SRCS_R) \
        $(SRCS_complex) \
        $(SRCS_C)

.KEEP_STATE:

all:    $(LIBS)

lint:   lintcheck
```

```
**********************************************************
    2735 Sun May  4 03:04:47 2014
new/usr/src/lib/libm/Makefile.libm.com
**********************************************************
```

```
   1 #
   2 # This file and its contents are supplied under the terms of the
   3 # Common Development and Distribution License ("CDDL"), version 1.0.
   4 # You may only use this file in accordance with the terms of version
   5 # 1.0 of the CDDL.
   6 #
   7 # A full copy of the text of the CDDL should have accompanied this
   8 # source.  A copy of the CDDL is also available via the Internet at
   9 # http://www.illumos.org/license/CDDL.
  10 #

  12 #
  13 # Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
  14 #

  16 LIBMDIR         = $(SRC)/lib/libm

  18 LIBMSRC         = $(LIBMDIR)/common

  20 CPP_CMD         = $(CC) -E -Xs

  22 ASSUFFIX_sparc  = S
  23 ASSUFFIX_i386   = s
  24 ASSUFFIX        = $(ASSUFFIX_$(MACH))

  26 # C99MODE of neither enabled nor disabled is "no_lib", whereby we expect
  27 # C99-the-language, but don't modify the behaviour of library routines.  This
  28 # is VERY IMPORTANT, as -xc99=%all, for instance, would link us with
  29 # values-xpg6, which would introduce an __xpg6 to our object with the C99
  30 # flags set, causing us to default C99 libm behaviour on, breaking
  31 # compatibility.
  32 C99MODE         =

  34 M4FLAGS         = -D__STDC__ -DELFOBJ -DPIC

  36 LDBLDIR_sparc   = Q
  37 LDBLDIR_i386    = LD
  38 LDBLDIR         = $(LDBLDIR_$(MACH))

  40 LM_IL           = $(LIBMDIR)/$(TARGET_ARCH)/src/locallibm.il

  42 CFLAGS          += $(C_PICFLAGS) -D__INLINE $(XSTRCONST) $(LM_IL)
  43 CFLAGS64        += $(C_PICFLAGS) -D__INLINE $(XSTRCONST) $(LM_IL)
  44 sparc_CFLAGS    += -Wa,-xarch=v8plus

  46 CDEF_i386       = -DCOMPARISON_MACRO_BUG
  46 CPPFLAGS        += -DELFOBJ \
  47                 -DLIBM_MT_FEX_SYNC \
  49                 $(CDEF_$(TARGET_ARCH)) \
  48                 -I$(LIBMSRC)/C \
  49                 -I$(LIBMSRC)/$(LDBLDIR) -I$(LIBMDIR)/$(TARGET_ARCH)/src

  51 # GCC needs __C99FEATURES__ such that the implementations of isunordered,
  52 # isgreaterequal, islessequal, etc, exist.  This is basically equivalent to
  53 # providing no -xc99 to Studio, in that it gets us the C99 language features,
  54 # but not values-xpg6, the reason for which is outline with C99MODE.
  55 CFLAGS          += -_gcc=-D__C99FEATURES__
  56 CFLAGS64        += -_gcc=-D__C99FEATURES__

  58 # libm depends on integer overflow characteristics
  59 CFLAGS          += -_gcc=-fno-strict-overflow
  60 CFLAGS64        += -_gcc=-fno-strict-overflow
```



```
  62 $(DYNLIB)       := LDLIBS += -lc

  64 $(LINTLIB)      := SRCS = $(LIBMSRC)/$(LINTSRC)

  66 CLEANFILES      += pics/*.s pics/*.S

  68 FPDEF_amd64     = -DARCH_amd64
  69 FPDEF_sparc     = -DCG89 -DARCH_v8plus -DFPADD_TRAPS_INCOMPLETE_ON_NAN
  70 FPDEF_sparcv9   = -DARCH_v9 -DFPADD_TRAPS_INCOMPLETE_ON_NAN
  71 FPDEF           = $(FPDEF_$(TARGET_ARCH))

  73 ASFLAGS         = -P -D_ASM $(FPDEF)

  75 XARCH_sparc     = v8plus
  76 XARCH_sparcv9   = v9
  77 XARCH_i386      = f80387
  78 XARCH_amd64     = amd64
  79 XARCH           = $(XARCH_$(TARGET_ARCH))

  81 ASOPT_sparc     = -xarch=$(XARCH) $(AS_PICFLAGS)
  82 ASOPT_sparcv9   = -xarch=$(XARCH) $(AS_PICFLAGS)
  83 ASOPT_i386      =
  84 ASOPT_amd64     = -xarch=$(XARCH) $(AS_PICFLAGS)
  85 ASOPT           = $(ASOPT_$(TARGET_ARCH))

  87 ASFLAGS         += $(ASOPT)

  89 CPPFLAGS_sparc = -DFPADD_TRAPS_INCOMPLETE_ON_NAN \
  90         -DFDTOS_TRAPS_INCOMPLETE_IN_FNS_MODE

  92 CPPFLAGS        += $(CPPFLAGS_$(MACH))
  93 ASFLAGS         += $(CPPFLAGS)
```

```
**********************************************************
   19733 Sun May  4 03:04:48 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-parentheses
516 CERRWARN        += -_gcc=-Wno-unused-variable

518 #endif /* ! codereview */
519 CPPFLAGS        += -DLIBM_BUILD

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

524 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 \
```

```
657         ../common/R/tanhf.c

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



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

730 SRCS_Q_sparc = \
731         $(SRCS_Q)

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

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

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

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

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

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

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



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

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

```
        ../common/C/expm1.c \
        ../common/C/floor.c \
        ../common/C/fmod.c \
        ../common/C/hypot.c \
        ../common/C/ilogb.c \
        ../common/C/isnan.c \
        ../common/C/log.c \
        ../common/C/log10.c \
        ../common/C/log2.c \
        ../common/C/pow.c \
        ../common/C/remainder.c \
        ../common/C/rint.c \
        ../common/C/scalbn.c \
        ../common/C/sin.c \
        ../common/C/sincos.c \
        ../common/C/tan.c

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

SRCS_sparc_sparc = \
        $(SRCS_i386_i386)

SRCS_sparc_sparcv9 = \
        ../common/C/copysign.c \
        ../common/C/fabs.c \
        ../common/C/nextafter.c

SRCS_i386_amd64 = \
        ../common/C/_TBL_atan.c \
        ../common/C/_TBL_exp2.c \
        ../common/C/_TBL_log.c \
        ../common/C/_TBL_log2.c \
        ../common/C/__tan.c \
        ../common/C/_TBL_tan.c \
        ../common/C/copysign.c \
        ../common/C/exp.c \
        ../common/C/fabs.c \
        ../common/C/ilogb.c \
        ../common/C/isnan.c \
        ../common/C/nextafter.c \
        ../common/C/rint.c \
        ../common/C/scalbn.c \
        ../common/C/acos.c \
        ../common/C/asin.c \
        ../common/C/atan.c \
        ../common/C/atan2.c \
        ../common/C/ceil.c \
        ../common/C/cos.c \
        ../common/C/exp10.c \
        ../common/C/exp2.c \
        ../common/C/expm1.c \
        ../common/C/floor.c \
        ../common/C/hypot.c \
        ../common/C/log.c \
        ../common/C/log10.c \
        ../common/C/log2.c \
        ../common/C/pow.c \
        ../common/C/sin.c \
        ../common/C/sincos.c \
        ../common/C/tan.c

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



```
        ../common/C/__lgamma.c \
        ../common/C/__rem_pio2.c \
        ../common/C/__rem_pio2m.c \
        ../common/C/__sin.c \
        ../common/C/__sincos.c \
        ../common/C/__xpg6.c \
        ../common/C/_lib_version.c \
        ../common/C/_SVID_error.c \
        ../common/C/_TBL_ipio2.c \
        ../common/C/_TBL_sin.c \
        ../common/C/acosh.c \
        ../common/C/asinh.c \
        ../common/C/atan2pi.c \
        ../common/C/atanh.c \
        ../common/C/cbrt.c \
        ../common/C/cosh.c \
        ../common/C/erf.c \
        ../common/C/gamma.c \
        ../common/C/gamma_r.c \
        ../common/C/j0.c \
        ../common/C/j1.c \
        ../common/C/jn.c \
        ../common/C/lgamma.c \
        ../common/C/lgamma_r.c \
        ../common/C/log1p.c \
        ../common/C/logb.c \
        ../common/C/matherr.c \
        ../common/C/scalb.c \
        ../common/C/signgam.c \
        ../common/C/significand.c \
        ../common/C/sincospi.c \
        ../common/C/sinh.c \
        ../common/C/sqrt.c \
        ../common/C/tanh.c

SRCS    = \
        $(SRCS_Q_$(MACH)) \
        $(SRCS_LD_$(MACH)) \
        $(SRCS_R) \
        $(SRCS_complex) \
        $(SRCS_C)

.KEEP_STATE:

all:    $(LIBS)

lint:   lintcheck
```

```
**********************************************************
    2735 Sun May  4 03:04:50 2014
new/usr/src/lib/libm/Makefile.libm.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

LIBMSRC         = $(LIBMDIR)/common

CPP_CMD         = $(CC) -E -Xs

ASSUFFIX_sparc  = S
ASSUFFIX_i386   = s
ASSUFFIX        = $(ASSUFFIX_$(MACH))

# C99MODE of neither enabled nor disabled is "no_lib", whereby we expect
# C99-the-language, but don't modify the behaviour of library routines.  This
# is VERY IMPORTANT, as -xc99=%all, for instance, would link us with
# values-xpg6, which would introduce an __xpg6 to our object with the C99
# flags set, causing us to default C99 libm behaviour on, breaking
# compatibility.
C99MODE         =

M4FLAGS         = -D__STDC__ -DELFOBJ -DPIC

LDBLDIR_sparc   = Q
LDBLDIR_i386    = LD
LDBLDIR         = $(LDBLDIR_$(MACH))

LM_IL           = $(LIBMDIR)/$(TARGET_ARCH)/src/locallibm.il

CFLAGS          += $(C_PICFLAGS) -D__INLINE $(XSTRCONST) $(LM_IL)
CFLAGS64        += $(C_PICFLAGS) -D__INLINE $(XSTRCONST) $(LM_IL)
sparc_CFLAGS    += -Wa,-xarch=v8plus

CDEF_i386       = -DCOMPARISON_MACRO_BUG
CPPFLAGS        += -DELFOBJ \
                -DLIBM_MT_FEX_SYNC \
                $(CDEF_$(TARGET_ARCH)) \
                -I$(LIBMSRC)/C \
                -I$(LIBMSRC)/$(LDBLDIR) -I$(LIBMDIR)/$(TARGET_ARCH)/src

# GCC needs __C99FEATURES__ such that the implementations of isunordered,
# isgreaterequal, islessequal, etc, exist.  This is basically equivalent to
# providing no -xc99 to Studio, in that it gets us the C99 language features,
# but not values-xpg6, the reason for which is outline with C99MODE.
CFLAGS          += -_gcc=-D__C99FEATURES__
CFLAGS64        += -_gcc=-D__C99FEATURES__

# libm depends on integer overflow characteristics
CFLAGS          += -_gcc=-fno-strict-overflow
CFLAGS64        += -_gcc=-fno-strict-overflow
```



```
$(DYNLIB)       := LDLIBS += -lc

$(LINTLIB)      := SRCS = $(LIBMSRC)/$(LINTSRC)

CLEANFILES      += pics/*.s pics/*.S

FPDEF_amd64     = -DARCH_amd64
FPDEF_sparc     = -DCG89 -DARCH_v8plus -DFPADD_TRAPS_INCOMPLETE_ON_NAN
FPDEF_sparcv9   = -DARCH_v9 -DFPADD_TRAPS_INCOMPLETE_ON_NAN
FPDEF           = $(FPDEF_$(TARGET_ARCH))

ASFLAGS         = -P -D_ASM $(FPDEF)

XARCH_sparc     = v8plus
XARCH_sparcv9   = v9
XARCH_i386      = f80387
XARCH_amd64     = amd64
XARCH           = $(XARCH_$(TARGET_ARCH))

ASOPT_sparc     = -xarch=$(XARCH) $(AS_PICFLAGS)
ASOPT_sparcv9   = -xarch=$(XARCH) $(AS_PICFLAGS)
ASOPT_i386      =
ASOPT_amd64     = -xarch=$(XARCH) $(AS_PICFLAGS)
ASOPT           = $(ASOPT_$(TARGET_ARCH))

ASFLAGS         += $(ASOPT)

CPPFLAGS_sparc = -DFPADD_TRAPS_INCOMPLETE_ON_NAN \
        -DFDTOS_TRAPS_INCOMPLETE_IN_FNS_MODE

CPPFLAGS        += $(CPPFLAGS_$(MACH))
ASFLAGS         += $(CPPFLAGS)
```

```
**********************************************************
    3415 Sun May  4 03:04:51 2014
new/usr/src/lib/libm/amd64/src/ieee_funcl.s
**********************************************************
______unchanged_portion_omitted_

  58         ENTRY(isnormall)
  59                                         / TRUE iff (x is finite, but
  60                                         /           neither subnormal nor zero)
  61                                         /      iff (msb(sgnfcnd(x) /= 0
  62                                         /           &  0 < bexp(x) < 0x7fff)
  63         movl    12(%rsp),%eax           / eax <-- hi_32(sgnfcnd(x))
  64         andl    $-0x80000000,%eax       / eax[31]  <-- msb(sgnfcnd(x)),
  64         movq    $0x80000000,%r8
  65         andq    %r8,%rax                / eax[31]  <-- msb(sgnfcnd(x)),
  65                                         / rest_of(eax) <-- 0
  66         jz      .L8                     / jump iff msb(sgnfcnd(x)) = 0
  67         movl    16(%rsp),%eax           / ax <-- sign and bexp of x
  68         notl    %eax                    / ax[0..14] <-- not(bexp(x))
  69         andq    $0x7fff,%rax            / eax  <-- zero_xtnd(not(bexp(x)))
  70         jz      .L8                     / jump  iff bexp(x) = 0x7fff or 0
  71         xorq    $0x7fff,%rax            / treat pseudo-denormal as subnormal
  72         jz      .L8
  73         movq    $1,%rax
  74 .L8:
  75         ret
  76         .align  16
  77         SET_SIZE(isnormall)

  79         ENTRY(issubnormall)
  80                                         / TRUE iff (bexp(x) = 0 &
  81                                         / msb(sgnfcnd(x)) = 0 & frac(x) /= 0)
  82         movl    12(%rsp),%eax           / eax <-- hi_32(sgnfcnd(x))
  83         testl   $0x80000000,%eax        / eax[31] = msb(sgnfcnd(x));
  84                                         / set ZF if it's 0.
  85                                         / set ZF if it is 0.
  85         jz      .may_be_subnorm         / jump iff msb(sgnfcnd(x)) = 0
  86 .not_subnorm:
  87         movq    $0,%rax
  88         ret
  89 .may_be_subnorm:
  90         testl   $0x7fff,16(%rsp)        / set ZF iff bexp(x)  = 0
  91         jnz     .not_subnorm            / jump   iff bexp(x) /= 0
  92         orl     8(%rsp),%eax            / (eax) = 0 iff sgnfcnd(x) = 0
  93         jz      .not_subnorm
  94         movq    $1,%rax
  95         ret
  96         .align  16
  97         SET_SIZE(issubnormall)
______unchanged_portion_omitted_
```

```
**********************************************************
    3962 Sun May  4 03:04:53 2014
new/usr/src/lib/libm/amd64/src/libm_inlines.h
**********************************************************
```

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

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

/*
 * Copyright 2011, Richard Lowe.
 */

/* Functions in this file are duplicated in locallibm.il.  Keep them in sync */
/* Functions in this file are duplicated in libm.m4.  Keep them in sync */

#ifndef _LIBM_INLINES_H
#define _LIBM_INLINES_H

#ifdef __GNUC__

#ifdef __cplusplus
extern "C" {
#endif

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

extern __inline__ double
__ieee754_sqrt(double a)
{
        double ret;

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

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



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

*unchanged_portion_omitted_*

```
extern __inline__ double
__ieee754_sqrt(double a)
/* XXX: Not actually called */
extern __inline__ short
__inline_fstsw(void)
{
        return (__inline_sqrt(a));
        short ret;

        __asm__ __volatile__("fstsw %0\n\t" : "=r" (ret));
        return (ret);
}
```

*unchanged_portion_omitted_*

```
extern __inline__ int
abs(int i)
{
        int ret;
        __asm__ __volatile__(
            "movl    %1, %0\n\t"
            "movl    %1,%0\n\t"
            "negl    %1\n\t"
            "cmovnsl %1, %0\n\t"
            : "=r" (ret), "+r" (i)
            :
            : "cc");
            "cmovnsl %1,%0\n\t"
            : "=r" (ret), "+r" (i));
        return (ret);
}

extern __inline__ double
copysign(double d1, double d2)
{
        double tmpd;
        double ret;

        __asm__ __volatile__(
            "movd %3, %1\n\t"
            "andpd %1, %0\n\t"
            "andnpd %2, %1\n\t"
            "orpd %1, %0\n\t"
            : "+x" (d1), "=x" (tmpd)
            : "x" (d2), "r" (0x7fffffffffffffff));
            "movq   $0x7fffffffffffffff,%%rax\n\t"
            "movd   %%rax,%%xmm2\n\t"
            "andpd  %%xmm2,%0\n\t"
            "andnpd %1,%%xmm2\n\t"
            "orpd   %%xmm2,%0\n\t"
            : "=x" (ret)
            : "x" (d2), "0" (d1)
            : "xmm2", "rax");

        return (d1);
        return (ret);
}

extern __inline__ double
d_sqrt_(double *d)
{
        double ret;
```

```
163         __asm__ __volatile__(
164             "movlpd %1,%0\n\t"
165             "sqrtsd %0,%0"
166             : "=x" (ret)
167             : "m" (*d));
168         return (ret);
144 }

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

151         __asm__ __volatile__(
152             "movd  %2, %1\n\t"
153             "andpd %1, %0"
154             : "+x" (d), "=x" (tmp)
155             : "r" (0x7fffffffffffffff));
177             "movq  $0x7fffffffffffffff,%%rax\n\t"
178             "movd  %%rax,%%xmm1\n\t"
179             "andpd %%xmm1,%0"
180             : "=x" (ret)
181             : "0" (d)
182             : "rax", "xmm1");

157         return (d);
184         return (ret);
158 }

160 extern __inline__ float
161 fabsf(float d)
162 {
190         float ret;

163         __asm__ __volatile__(
164             "andpd %1, %0"
165             : "+x" (d)
166             : "x" (0x7fffffff));
193             "andpd %2,%0"
194             : "=x" (ret)
195             : "0" (d), "x" (0x7fffffff));

168         return (d);
197         return (ret);
169 }

171 extern __inline__ int
172 finite(double d)
173 {
174     long ret = 0x7fffffffffffffff;
175         uint64_t tmp;
203         long ret;                   /* A long, so gcc chooses an %r* for %0 */

177         __asm__ __volatile__(
178             "movq %2, %1\n\t"
179             "andq %1, %0\n\t"
180             "movq $0x7ff0000000000000, %1\n\t"
181             "subq %1, %0\n\t"
182             "shrq $63, %0\n\t"
183             : "+r" (ret), "=r" (tmp)
206             "movq %1,%%rcx\n\t"
207             "movq $0x7fffffffffffffff,%0\n\t"
208             "andq %%rcx,%0\n\t"
209             "movq $0x7ff0000000000000,%%rcx\n\t"
210             "subq %%rcx,%0\n\t"
```



```
211             "shrq $63,%0\n\t"
212             : "=r" (ret)
184             : "x" (d)
185             : "cc");
214             : "rcx");

216         return (ret);
217 }

219 extern __inline__ float
220 r_sqrt_(float *f)
221 {
222         float ret;

224         __asm__ __volatile__(
225             "movss  %1,%0\n\t"
226             "sqrtss %0,%0\n\t"
227             : "+x" (ret)
228             : "m" (*f));
187         return (ret);
188 }

190 extern __inline__ int
191 signbit(double d)
192 {
193         long ret;
194         __asm__ __volatile__(
195             "movmskpd %1, %0\n\t"
237             "movmskpd %1,%0\n\t"
196             "andq     $1, %0\n\t"
197             : "=r" (ret)
198             : "x" (d)
199             : "cc");
240             : "x" (d));
241         return (ret);
242 }

244 extern __inline__ int
245 signbitf(float f)
246 {
247         int ret;
248         __asm__ __volatile__(
249             "movskps %1,%0\n\t"
250             "andq    $1, %0\n\t"
251             : "=r" (ret)
252             : "x" (f));
200         return (ret);
201 }

203 extern __inline__ double
204 sqrt(double d)
205 {
206         return (__inline_sqrt(d));
259         double ret;

261         __asm__ __volatile__(
262             "sqrtsd %0, %0"
263             : "=x" (ret)
264             : "0" (d));
265         return (ret);
207 }

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

```
271         float ret;
273         __asm__ __volatile__(
274             "sqrtss %0, %0"
275             : "=x" (ret)
276             : "0" (f));
277         return (ret);
213 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
    3242 Sun May  4 03:04:55 2014
new/usr/src/lib/libm/amd64/src/locallibm.il
**********************************************************
/
/ CDDL HEADER START
/
/ The contents of this file are subject to the terms of the
/ Common Development and Distribution License (the "License").
/ You may not use this file except in compliance with the License.
/
/ You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
/ or http://www.opensolaris.org/os/licensing.
/ See the License for the specific language governing permissions
/ and limitations under the License.
/
/ When distributing Covered Code,  this CDDL HEADER in each
/ file and  the License file at usr/src/OPENSOLARIS.LICENSE.
/ If applicable, add the following below this CDDL HEADER, with the
/ fields enclosed by brackets "[]" 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.
/

/ Portions of this file are duplicated as GCC inline assembly in
/ libm_inlines.h.  Keep them in sync.

        .inline __ieee754_sqrt,0
        sqrtsd  %xmm0,%xmm0
        .end

        .inline __inline_sqrtf,0
        sqrtss  %xmm0,%xmm0
        .end

        .inline __inline_sqrt,0
        sqrtsd  %xmm0,%xmm0
        .end

        .inline __inline_fstsw,0
        fstsw   %ax
        .end

/
/ 00 - 24 bits
/ 01 - reserved
/ 10 - 53 bits
/ 11 - 64 bits
/
        .inline __swapRP,0
        subq    $16,%rsp
        fstcw   (%rsp)
        movw    (%rsp),%ax
        movw    %ax,%cx
        andw    $0xfcff,%cx
        andl    $0x3,%edi
        shlw    $8,%di
        orw     %di,%cx
        movl    %ecx,(%rsp)
```



```
        fldcw   (%rsp)
        shrw    $8,%ax
        andq    $0x3,%rax
        addq    $16,%rsp
        .end

/
/ 00 - Round to nearest, with even preferred
/ 01 - Round down
/ 10 - Round up
/ 11 - Chop
/
        .inline __swap87RD,0
        subq    $16,%rsp
        fstcw   (%rsp)
        movw    (%rsp),%ax
        movw    %ax,%cx
        andw    $0xf3ff,%cx
        andl    $0x3,%edi
        shlw    $10,%di
        orw     %di,%cx
        movl    %ecx,(%rsp)
        fldcw   (%rsp)
        shrw    $10,%ax
        andq    $0x3,%rax
        addq    $16,%rsp
        .end

        .inline abs,0
        cmpl    $0,%edi
        jge     1f
        negl    %edi
1:      movl    %edi,%eax
        .end

        .inline __copysign,0
        movq    $0x7fffffffffffffff,%rax
        movdq   %rax,%xmm2
        andpd   %xmm2,%xmm0
        andnpd  %xmm1,%xmm2
        orpd    %xmm2,%xmm0
        .end

        .inline __d_sqrt_,0
        movlpd  (%rdi),%xmm0
        sqrtsd  %xmm0,%xmm0
        .end

        .inline __fabs,0
        movq    $0x7fffffffffffffff,%rax
        movdq   %rax,%xmm1
        andpd   %xmm1,%xmm0
        .end

        .inline __fabsf,0
        movl    $0x7fffffff,%eax
        movdl   %eax,%xmm1
        andps   %xmm1,%xmm0
        .end

        .inline _finite,0
        subq    $16,%rsp
        movlpd  %xmm0,(%rsp)
        movq    (%rsp),%rcx
        movq    $0x7fffffffffffffff,%rax
        andq    %rcx,%rax
```

```
120         movq    $0x7ff0000000000000,%rcx
121         subq    %rcx,%rax
122         shrq    $63,%rax
123         addq    $16,%rsp
124         .end

135         .inline __r_sqrt_,0
136         movss   (%rdi),%xmm0
137         sqrtss  %xmm0,%xmm0
138         .end

126         .inline __signbit,0
127         movmskpd %xmm0,%eax
142         andq    $1,%rax
143         .end

145         .inline __signbitf,0
146         movmskps %xmm0,%eax
128         andq    $1,%rax
129         .end

131         .inline __sqrt,0
132         sqrtsd  %xmm0,%xmm0
133         .end

135         .inline __sqrtf,0
136         sqrtss  %xmm0,%xmm0
137         .end

139         .inline __f95_signf,0
140         movl    (%rdi),%eax
141         movl    (%rsi),%ecx
142         andl    $0x7fffffff,%eax
143         andl    $0x80000000,%ecx
144         orl     %ecx,%eax
145         movdl   %eax,%xmm0
146         .end

148         .inline __f95_sign,0
149         movq    (%rsi),%rax
150         movq    $0x7fffffffffffffff,%rdx
151         shrq    $63,%rax
152         shlq    $63,%rax
153         andq    (%rdi),%rdx
154         orq     %rdx,%rax
155         movdq   %rax,%xmm0
156         .end

158         .inline __r_sign,0
159         movl    $0x7fffffff,%eax
160         movl    $0x80000000,%edx
161         andl    (%rdi),%eax
162         cmpl    (%rsi),%edx
163         cmovel  %eax,%edx
164         andl    (%rsi),%edx
165         orl     %edx,%eax
166         movdl   %eax,%xmm0
167         .end

169         .inline __d_sign,0
170         movq    $0x7fffffffffffffff,%rax
171         movq    $0x8000000000000000,%rdx
172         andq    (%rdi),%rax
173         cmpq    (%rsi),%rdx
174         cmoveq  %rax,%rdx
175         andq    (%rsi),%rdx
```



```
176         orq     %rdx,%rax
177         movdq   %rax,%xmm0
178         .end
```

****
22354 Sun May 4 03:04:56 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  4 03:04:58 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  4 03:05:00 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  4 03:05:02 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  4 03:05:04 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  4 03:05:06 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  4 03:05:07 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  4 03:05:08 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 }
```

```
**********************************************************
    4615 Sun May  4 03:05:10 2014
new/usr/src/lib/libm/common/LD/__cosl.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_cosl( long double x;  long double y )
 * kernel cos function on [-pi/4, pi/4], pi/4 ~ 0.785398164
 * Input x is assumed to be bounded by ~pi/4 in magnitude.
 * Input y is the tail of x.
 *
 * Table look up algorithm
 *      1. by cos(-x) = cos(x), we may replace x by |x|
 *      2. if x < 25/128 = [0x3ffc4000, 0] = 0.15625 , then
 *           if x < 2^-57 (hx < 0x3fc60000 0), return 1.0 with inexact if x!= 0
 *           z = x*x;
 *           if x <= 1/128 = 2**-7 = 0.0078125
 *              cos(x)=1.0+z*(qq1+z*(qq2+z*(qq3+z*(qq4+z*qq5))))
 *           else
 *              cos(x)=1.0+z*(q1+ ... z*q8)
 *      3. else
 *              ht = (hx + 0x400)&0x7ffff800    (round x to a break point t)
 *              lt = 0
 *              i  = (hy-0x3ffc4000)>>11;       (i<=64)
 *              x' = (x - t)+y                  (|x'| ~<= 2^-7
 *         By
 *              cos(t+x')
 *                = cos(t)cos(x')-sin(t)sin(x')
 *                = cos(t)(1+z*(qq1+z*qq2))-[sin(t)]*x*(1+z*(pp1+z*pp2))
 *                = cos(t) + [cos(t)]*(z*(qq1+z*qq2))-
 *                              [sin(t)]*x*(1+z*(pp1+z*pp2))
 *
 *         Thus,
 *              let a= _TBL_cos_hi[i], b = _TBL_cos_lo[i], c= _TBL_sin_hi[i],
 *              x = (x-t)+y
 *              z = x*x;
 *              cos(t+x) = a+(b+ (-c*x*(1+z*(pp1+z*pp2))+a*(z*(qq1+z*qq2)))
```



```
 */

#include "libm.h"

#include <sys/isa_defs.h>

extern const long double _TBL_cosl_hi[], _TBL_cosl_lo[], _TBL_sinl_hi[];
static const long double
one     = 1.0,
/*
 * |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,
/*
 *
 * |cos(x) - (1+q1*x^2+...+q8*x^16)| <= 2^-117.11 for |x|<= 0.15625
 */
q1      =  -4.99999999999999999999999999999756416975e-0001L,
q2      =   4.16666666666666666666666664006066577258e-0002L,
q3      =  -1.38888888888888888888887770036393716963 7e-0003L,
q4      =   2.48015873015873015849446846303181408355 9e-0005L,
q5      =  -2.75573192239858627632281925035600554287 1e-0007L,
q6      =   2.08767569876742426144195976072985401785 5e-0009L,
q7      =  -1.14707448123966208907245212901079077476 1e-0011L,
q8      =   4.77776164739965159973066342226353103478 2e-0014L,
/*
 *
 * |cos(x) - (1+qq1*x^2+...+ qq5*x^10)| <= 2^-123.84 for |x|<=1/128
 */
qq1     =  -4.99999999999999999999999999999378373641e-0001L,
qq2     =   4.16666666666666666666665478399327703130e-0002L,
qq3     =  -1.38888888888888888805821123061805161349 4e-0003L,
qq4     =   2.48015873015610537777158565890530311186 6e-0005L,
qq5     =  -2.75572809976252632573648837669515700873 6e-0007L;
/* INDENT ON */
long double
__k_cosl(long double x, long double y) {
        long double a, t, z, w;
        int *pt = (int *) &t, *px = (int *) &x;
        int i, j, hx, ix;

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

                if (ix < 0x3ff80000)    /* 0.0078125 */
                        return (one + z * (qq1 + z * (qq2 + z * (qq3 + z *
                                (qq4 + z * qq5)))));
                else
                        return (one + z * (q1 + z * (q2 + z * (q3 + z * (q4 +
```

```
126                                 z * (q5 + z * (q6 + z * (q7 + z * q8))))))));
127         }
128         j = (ix + 0x400) & 0x7ffff800;
129         i = (j - 0x3ffc4000) >> 11;
130 #if defined(__i386) || defined(__amd64)
131         ITOX(j, pt);
132 #else
130 #if defined(_BIG_ENDIAN)
133         pt[0] = j;
132 #else
133         ITOX(j, pt);
134 #endif
135         if (hx > 0)
136                 x = y - (t - x);
137         else
138                 x = (-y) - (t + x);
139         a = _TBL_cosl_hi[i];
140         z = x * x;
141         t = z * (qq1 + z * (qq2 + z * (qq3 + z * (qq4 + z * qq5))));
142         w = x * (one + z * (pp1 + z * (pp2 + z * (pp3 + z * (pp4 + z *
143                 pp5)))));
144         t = _TBL_cosl_lo[i] - (_TBL_sinl_hi[i] * w - a * t);
145         return (a + t);
146 }
```

*unchanged_portion_omitted_*

```
**********************************************************
    1935 Sun May  4 03:05:12 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"
#endif /* ! codereview */

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);
}
```

```
**********************************************************
    4822 Sun May  4 03:05:14 2014
new/usr/src/lib/libm/common/LD/__sincosl.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 */
/*
 * long double __k_sincos( long double x, long double y, long double *c )
 * kernel sincosl function on [-pi/4, pi/4], pi/4 ~ 0.785398164
 * Input x is assumed to be bounded by ~pi/4 in magnitude.
 * Input y is the tail of x.
 * return sinl(x) with *c = cosl(x)
 *
 * Table look up algorithm
 *      see __k_sinl() and __k_cosl()
 */

#include "libm.h"

#include <sys/isa_defs.h>

extern const long double _TBL_sinl_hi[], _TBL_sinl_lo[], _TBL_cosl_hi[],
        _TBL_cosl_lo[];
static const long double
one     = 1.0,
/*
 * |sin(x) - (x+pp1*x^3+...+pp5*x^11)| <= 2^-122.32 for |x|<1/64
 */
pp1     = -1.6666666666666666666666666665586782940810e-0001L,
pp2     =  8.3333333333333333333300372366092931754 0e-0003L,
pp3     = -1.9841269841269840760459034837783378 37804470e-0004L,
pp4     =  2.7557319223619066413197231062109009494 13e-0006L,
pp5     = -2.5051983985709470190939984691350120576 73e-0008L,
/*
 * |(sin(x) - (x+p1*x^3+...+p8*x^17)|
 * |------------------------------- | <= 2^-116.17 for |x|<0.1953125
 * |                x              |
 */
```



```
p1      =  -1.6666666666666666666666666666211262297e-0001L,
p2      =   8.3333333333333333333333333301497876908541e-0003L,
p3      =  -1.9841269841269841269840413028811806219 22e-0004L,
p4      =   2.7557319223985890641005873513072696210 93e-0006L,
p5      =  -2.5052108385441631293789069537655953938 73e-0008L,
p6      =   1.6059043836432443750509982437785340742 73e-0010L,
p7      =  -7.6471627228006855169014561142708246226 99e-0013L,
p8      =   2.8100464286619029617254288410688444626 03e-0015L,
/*
 *
 * |cos(x) - (1+qq1*x^2+...+ qq5*x^10)| <= 2^-123.84 for |x|<=1/128
 */
qq1     =  -4.9999999999999999999999999999378373641e-0001L,
qq2     =   4.1666666666666666666666654783993277031 30e-0002L,
qq3     =  -1.3888888888888888888058211230618051613 494e-0003L,
qq4     =   2.4801587301561053777715856589053031118 66e-0005L,
qq5     =  -2.7557280997625263257364883766951570087 36e-0007L,
/*
 *
 * |cos(x) - (1+q1*x^2+...+ q8*x^16)| <= 2^-117.11 for |x|<= 0.15625
 */
q1      =  -4.9999999999999999999999999999756416975e-0001L,
q2      =   4.1666666666666666666666666404006657725 8e-0002L,
q3      =  -1.3888888888888888888888777003639371696 37e-0003L,
q4      =   2.4801587301587301584944684630318140835 59e-0005L,
q5      =  -2.7557319223985862763228192503560055428 71e-0007L,
q6      =   2.0876756987674242614419597607298540178 55e-0009L,
q7      =  -1.1470744812396620890724521290107907747 61e-0011L,
q8      =   4.7777616473996515997306634222635310347 82e-0014L;
/* INDENT ON */
long double
__k_sincosl(long double x, long double y, long double *c) {
        long double a1, a2, t, t1, t2, z, w;
        int *pt = (int *) &t, *px = (int *) &x;
        int i, j, hx, ix;

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

                if (ix < 0x3ff80000) {
                        *c = one + z * (qq1 + z * (qq2 + z * (qq3 + z * (qq4 +
                                z * qq5))));
                        t = z * (p1 + z * (p2 + z * (p3 + z * (p4 + z * (p5 +
                                z * p6)))));
                } else {
                        *c = one + z * (q1 + z * (q2 + z * (q3 + z * (q4 + z *
                                (q5 + z * (q6 + z * (q7 + z * q8)))))));
                        t = z * (p1 + z * (p2 + z * (p3 + z * (p4 + z * (p5 +
                                z * (p6 + z * (p7 + z * p8)))))));
                }
```

```
126                 t = y + x * t;
127                 return (x + t);
128         }
129         j = (ix + 0x400) & 0x7ffff800;
130         i = (j - 0x3ffc4000) >> 11;
131 #if defined(__i386) || defined(__amd64)
132         ITOX(j, pt);
133 #else
131 #if defined(_BIG_ENDIAN)
134         pt[0] = j;
133 #else
134         ITOX(j, pt);
135 #endif
136         if (hx > 0)
137                 x = y - (t - x);
138         else
139                 x = (-y) - (t + x);
140         a1 = _TBL_sinl_hi[i];
141         z = x * x;
142         t = z * (qq1 + z * (qq2 + z * (qq3 + z * (qq4 + z * qq5))));
143         w = x * (one + z * (pp1 + z * (pp2 + z * (pp3 + z * (pp4 + z *
144                 pp5)))));
145         a2 = _TBL_cosl_hi[i];
146         t2 = _TBL_cosl_lo[i] - (a1 * w - a2 * t);
147         *c = a2 + t2;
148         t1 = a2 * w + a1 * t;
149         t1 += _TBL_sinl_lo[i];
150         if (hx < 0)
151                 return (-a1 - t1);
152         else
153                 return (a1 + t1);
154 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
    4646 Sun May  4 03:05:15 2014
new/usr/src/lib/libm/common/LD/__sinl.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_sinl( long double x;  long double y )
 * kernel sin function on [-pi/4, pi/4], pi/4 ~ 0.785398164
 * Input x is assumed to be bounded by ~pi/4 in magnitude.
 * Input y is the tail of x.
 *
 * Table look up algorithm
 *      1. by sin(-x) = -sin(x), need only to consider positive x
 *      2. if x < 25/128 = [0x3ffc9000,0,0,0] = 0.1953125 , then
 *           if x < 2^-57 (hx < 0x3fc60000,0,0,0), return x (inexact if x!= 0)
 *           z = x*x;
 *           if x <= 1/64 = 2**-6
 *              sin(x) = x + (y+(x*z)*(p1 + z*p2))
 *           else
 *              sin(x) = x + (y+(x*z)*(p1 + z*(p2 + z*(p3 + z*p4))))
 *      3. else
 *              ht = (hx + 0x400)&0x7ffff800    (round x to a break point t)
 *              lt = 0
 *              i  = (hy-0x3ffc4000)>>11;       (i<=64)
 *              x' = (x - t)+y                  (|x'| ~<= 2^-7
 *         By
 *              sin(t+x')
 *                = sin(t)cos(x')+cos(t)sin(x')
 *                = sin(t)(1+z*(qq1+z*qq2))+[cos(t)]*x*(1+z*(pp1+z*pp2))
 *                = sin(t) + [sin(t)]*(z*(qq1+z*qq2))+
 *                              [cos(t)]*x*(1+z*(pp1+z*pp2))
 *
 *         Thus,
 *              let a= _TBL_sin_hi[i], b = _TBL_sin_lo[i], c= _TBL_cos_hi[i],
 *              x = (x-t)+y
 *              z = x*x;
 *              sin(t+x) = a+(b+ ((c*x)*(1+z*(pp1+z*pp2))+a*(z*(qq1+z*qq2)))
```



```
 */

#include "libm.h"

#include <sys/isa_defs.h>

extern const long double _TBL_sinl_hi[], _TBL_sinl_lo[], _TBL_cosl_hi[];
static const long double
one     = 1.0,
/*
 * |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,
/*
 * |(sin(x) - (x+p1*x^3+...+p8*x^17)|
 * |------------------------------- | <= 2^-116.17 for |x|<0.1953125
 * |                x               |
 */
p1      = -1.66666666666666666666666666666211262297e-0001L,
p2      =  8.33333333333333333333333301497876908541e-0003L,
p3      = -1.98412698412698412698404130288118062192 2e-0004L,
p4      =  2.75573192239858906410058735130726962109 3e-0006L,
p5      = -2.50521083854416312937890695376559539387 3e-0008L,
p6      =  1.60590438364324437505099824377853407427 3e-0010L,
p7      = -7.64716272280068551690145611427082462269 9e-0013L,
p8      =  2.81004642866190296172542884106884446260 3e-0015L,
/*
 *                    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;
/* INDENT ON */
long double
__k_sinl(long double x, long double y) {
        long double a, t, z, w;
        int *pt = (int *) &t, *px = (int *) &x;
        int i, j, hx, ix;

        t = 1.0L;
#if defined(__i386) || defined(__amd64)
        XTOI(px, hx);
#else
#if defined(_BIG_ENDIAN)
        hx = px[0];
#else
        XTOI(px, hx);
#endif
        ix = hx & 0x7fffffff;
        if (ix < 0x3ffc9000) {
                if (ix < 0x3fc60000)
                        if (((int) x) == 0)
                                return (x);     /* generate inexact */
                z = x * x;
                t = z * (p1 + z * (p2 + z * (p3 + z * (p4 + z * (p5 + z *
                        (p6 + z * (p7 + z * p8)))))));
                t = y + x * t;
                return (x + t);
        }
```

```
126         j = (ix + 0x400) & 0x7ffff800;
127         i = (j - 0x3ffc4000) >> 11;
128 #if defined(__i386) || defined(__amd64)
129         ITOX(j, pt);
130 #else
128 #if defined(_BIG_ENDIAN)
131         pt[0] = j;
130 #else
131         ITOX(j, pt);
132 #endif
133         if (hx > 0)
134                 x = y - (t - x);
135         else
136                 x = (-y) - (t + x);
137         a = _TBL_sinl_hi[i];
138         z = x * x;
139         t = z * (qq1 + z * (qq2 + z * (qq3 + z * (qq4 + z * qq5))));
140         w = x * (one + z * (pp1 + z * (pp2 + z * (pp3 + z * (pp4 + z *
141                 pp5)))));
142         t = _TBL_cosl_hi[i] * w + a * t;
143         t += _TBL_sinl_lo[i];
144         if (hx < 0)
145                 return (-a - t);
146         else
147                 return (a + t);
148 }
```

*_____unchanged_portion_omitted_*

```
**********************************************************
    5472 Sun May  4 03:05:17 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
#if defined(_BIG_ENDIAN)
        hx = px[0];
#else
        XTOI(px, hx);
#endif
        ix = hx & 0x7fffffff;
```

```
125         if (ix < 0x3ffc4000) {
126                 if (ix < 0x3fc60000) {
127                         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
145 #if defined(_BIG_ENDIAN)
148         pt[0] = j;
147 #else
148         ITOX(j, pt);
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  4 03:05:18 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));
}
```

```
**********************************************************
    1779 Sun May  4 03:05:20 2014
new/usr/src/lib/libm/common/LD/cbrtl.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 cbrtl = __cbrtl
#endif

#include "libm.h"
#include "longdouble.h"
#endif /* ! codereview */

static const double d_one = 1.0;

long double
cbrtl(long double x) {
        long double s, t, r, w, y;
        double dx, dy;
        int *py = (int *) &dy;
        int n, m, m3, n0, sx;

        if (!finitel(x))
                return (x + x);
        if (iszerol(x))
                return (x);
        n0 = 0;
        if (*((int *) &d_one) == 0)
                n0 = 1;
        sx = signbitl(x);
        x = fabsl(x);
        n = ilogbl(x);
        m = n / 3;
        m3 = m + m + m;
        y = scalbnl(x, -m3);
        dx = (double) y;
        dy = cbrt(dx);
        py[1 - n0] += 2;
```



```
        if (py[1 - n0] == 0)
                py[n0] += 1;

        /* one step newton iteration to 113 bits with error < 0.667ulps */
        t = (long double) dy;
        t = scalbnl(t, m);
        s = t * t;
        r = x / s;
        w = t + t;
        r = (r - t) / (w + r);
        t += t * r;

        return (sx == 0 ? t : -t);
}
```

```
**********************************************************
    2815 Sun May  4 03:05:22 2014
new/usr/src/lib/libm/common/LD/coshl.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 coshl = __coshl
#endif

#include "libm.h"
#include "longdouble.h"
#endif /* ! codereview */

/*
 * COSH(X)
 * RETURN THE HYPERBOLIC COSINE OF X
 *
 * Method :
 *      1. Replace x by |x| (COSH(x) = COSH(-x)).
 *      2.
 *                                                  [ EXP(x) - 1 ]^2
 *          0       <= x <= 0.3465  :  COSH(x) := 1 + -------------------
 *                                                     2*EXP(x)
 *
 *                                               EXP(x) +  1/EXP(x)
 *          0.3465  <= x <= thresh  :  COSH(x) := -------------------
 *                                                        2
 *          thresh  <= x <= lnovft  :  COSH(x) := EXP(x)/2
 *          lnovft  <= x <  INF     :  COSH(x) := SCALBN(EXP(x-MEP1*ln2),ME)
 *
 *
 * here
 *      0.3465          a number that is near one half of ln2.
 *      thresh          a number such that
 *                              EXP(thresh)+EXP(-thresh)=EXP(thresh)
 *      lnovft          logarithm of the overflow threshold
 *                      = MEP1*ln2 chopped to machine precision.
 *      ME              maximum exponent
```



```
 *      MEP1            maximum exponent plus 1
 *
 * Special cases:
 *      COSH(x) is |x| if x is +INF, -INF, or NaN.
 *      only COSH(0)=1 is exact for finite x.
 */

static const long double C[] = {
        0.5L,
        1.0L,
        0.3465L,
        45.0L,
        1.13565234062941439487914 9e+04L,
        7.00444768624254908785898 5e-16L,
        2.71050543121376108501863 2e-20L,               /* 2^-65 */
};

#define half    C[0]
#define one     C[1]
#define thr1    C[2]
#define thr2    C[3]
#define lnovft  C[4]
#define lnovlo  C[5]
#define tinyl   C[6]

long double
coshl(long double x) {
        long double w, t;

        w = fabsl(x);
        if (!finitel(w))
                return (w + w); /* x is INF or NaN */
        if (w < thr1) {
                if (w < tinyl)
                        return (one + w);       /* inexact+directed rounding */
                t = expm1l(w);
                w = one + t;
                w = one + (t * t) / (w + w);
                return (w);
        }
        if (w < thr2) {
                t = expl(w);
                return (half * (t + one / t));
        }
        if (w <= lnovft)
                return (half * expl(w));
        return (scalbnl(expl((w - lnovft) - lnovlo), 16383));
}
```

```
**********************************************************
    2816 Sun May  4 03:05:24 2014
new/usr/src/lib/libm/common/LD/cosl.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 cosl = __cosl

/* INDENT OFF */
/* cosl(x)
 * Table look-up algorithm by K.C. Ng, November, 1989.
 *
 * kernel function:
 *      __k_sinl        ... sin function on [-pi/4,pi/4]
 *      __k_cosl        ... cos function on [-pi/4,pi/4]
 *      __rem_pio2l     ... argument reduction routine
 *
 * Method.
 *      Let S and C denote the sin and cos respectively on [-PI/4, +PI/4].
 *      1. Assume the argument x is reduced to y1+y2 = x-k*pi/2 in
 *         [-pi/2 , +pi/2], and let n = k mod 4.
 *      2. Let S=S(y1+y2), C=C(y1+y2). Depending on n, we have
 *
 *          n        sin(x)      cos(x)        tan(x)
 *     ----------------------------------------------------------
 *          0          S           C             S/C
 *          1          C          -S            -C/S
 *          2         -S          -C             S/C
 *          3         -C           S            -C/S
 *     ----------------------------------------------------------
 *
 * Special cases:
 *      Let trig be any of sin, cos, or tan.
 *      trig(+-INF)  is NaN, with signals;
 *      trig(NaN)    is that NaN;
 *
 * Accuracy:
 *      computer TRIG(x) returns trig(x) nearly rounded.
 */
```



```
/* INDENT ON */

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

#include <sys/isa_defs.h>

long double
cosl(long double x) {
        long double y[2], z = 0.0L;
        int n, ix;
        int *px = (int *) &x;

        /* trig(Inf or NaN) is NaN */
        if (!finitel(x))
                return x - x;

        /* High word of x. */
#if defined(__i386) || defined(__amd64)
        XTOI(px, ix);
#else
#if defined(_BIG_ENDIAN)
        ix = px[0];
#else
        XTOI(px, ix);
#endif

        /* |x| ~< pi/4 */
        ix &= 0x7fffffff;
        if (ix <= 0x3ffe9220)
                return __k_cosl(x, z);

        /* argument reduction needed */
        else {
                n = __rem_pio2l(x, y);
                switch (n & 3) {
                case 0:
                        return __k_cosl(y[0], y[1]);
                case 1:
                        return -__k_sinl(y[0], y[1]);
                case 2:
                        return -__k_cosl(y[0], y[1]);
                case 3:
                        return __k_sinl(y[0], y[1]);
                /* NOTREACHED */
                }
        }
    return 0.0L;
}
```

*_____unchanged_portion_omitted_*

```
**********************************************************
    1612 Sun May  4 03:05:25 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_
```

```
**********************************************************
   27705 Sun May  4 03:05:27 2014
new/usr/src/lib/libm/common/LD/j0l.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.
 */

/*
 * Floating point Bessel's function of the first and second kinds
 * of order zero: j0(x),y0(x);
 *
 * Special cases:
 *      y0(0)=y1(0)=yn(n,0) = -inf with division by zero signal;
 *      y0(-ve)=y1(-ve)=yn(n,-ve) are NaN with invalid signal.
 */

#pragma weak j0l = __j0l
#pragma weak y0l = __y0l

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

#endif /* ! codereview */
#include <math.h>
#if defined(__SUNPRO_C)
#include <sunmath.h>
#endif

#define GENERIC long double
static GENERIC
zero    = 0.0L,
small   = 1.0e-9L,
tiny    = 1.0e-38L,
one     = 1.0L,
five    = 5.0L,
eight   = 8.0L,
invsqrtpi= 5.641895835477562869480794515607725858441e-0001L,
tpi     = 0.636619772367581343075535053490057448L;
```



```
static GENERIC pzero(), qzero();
static GENERIC r0[7] = {
  -2.4999999999999999999999999999998934492e-0001L,
   1.2726579273600497863276184511337637148880e-0002L,
  -2.6944997637129632769006366934006596008980e-0004L,
   2.7248774750589775769032340709196164478830e-0006L,
  -1.4326171032143302369674774953930763202810e-0008L,
   3.8232488040800791687066835405137922244710e-0011L,
  -4.1831742775679836473375685042863136650650e-0014L,
};
static GENERIC s0[7] = {
   1.0e0L,
   1.1593682905598008546895261954628846663950e-0002L,
   6.6293975973949733830097438761699467725590e-0005L,
   2.4267799813940544063054311425017350943400e-0007L,
   6.0976634912485110690944004696354497498830e-0010L,
   1.0170191333409292202387474132160522240360e-0012L,
   9.0125931793061975795183745819693712784810e-0016L,
};

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

        if(isnanl(x)) return x+x;
        x = fabsl(x);
        if(x > 1.28L){
                if(!finitel(x)) return zero;
                s = sinl(x);
                c = cosl(x);
        /* j0(x) = sqrt(2/(pi*x))*(p0(x)*cos(x0)-q0(x)*sin(x0))
         * where x0 = x-pi/4
         *      Better formula:
         *              cos(x0) = cos(x)cos(pi/4)+sin(x)sin(pi/4)
         *                      = 1/sqrt(2) * (cos(x) + sin(x))
         *              sin(x0) = sin(x)cos(pi/4)-cos(x)sin(pi/4)
         *                      = 1/sqrt(2) * (sin(x) - cos(x))
         * To avoid cancellation, use
         *              sin(x) +- cos(x) = -cos(2x)/(sin(x) -+ cos(x))
         * to compute the worse one.
         */
                if(x>1.0e2450L) {       /* x+x may overflow */
                        ss = s-c;
                        cc = s+c;
                } else if(signbitl(s)!=signbitl(c)) {
                        ss = s - c;
                        cc = -cosl(x+x)/ss;
                } else {
                        cc = s + c;
                        ss = -cosl(x+x)/cc;
                }
        /*
         * j0(x) = 1/sqrt(pi) * (P(0,x)*cc - Q(0,x)*ss) / sqrt(x)
         * y0(x) = 1/sqrt(pi) * (P(0,x)*ss + Q(0,x)*cc) / sqrt(x)
         */
                if(x>1.0e120L) return (invsqrtpi*cc)/sqrtl(x);
                u = pzero(x); v = qzero(x);
                return invsqrtpi*(u*cc-v*ss)/sqrtl(x);
        }
        if(x<=small) {
            if(x<=tiny) return one-x;
            else return one-x*x*0.25L;
        }
        z = x*x;
```

```
129         r = r0[6]; s = s0[6];
130         for(i=5;i>=0;i--) {
131             r = r*z + r0[i];
132             s = s*z + s0[i];
133         }
134         return(one+z*(r/s));
135 }

137 static GENERIC u0[8] = {
138   -7.38042951086872252743439279484830163122 0e-0002L,
139    1.76685555962594079185753694930198181651 3e-0001L,
140   -1.38647072270104792323555325124016283940 8e-0002L,
141    3.52014924272481157863697081163122486261 5e-0004L,
142   -3.97859966324379004985364227562495187002 5e-0006L,
143    2.22880115326395722454722255680691547976 3e-0008L,
144   -6.12124676429878501865859717949883731617 7e-0011L,
145    6.67710362972267883347596581052558739659 6e-0014L,
146 };
147 static GENERIC v0[8] = {
148    1.0e0L,
149    1.24716441653911131157167676612776712797 0e-0002L,
150    7.82914474963979150005290028148936744357 6e-0005L,
151    3.24712654042224533051121832101336033660 6e-0007L,
152    9.75051672478949967856706257254956844786 9e-0010L,
153    2.15671322317359121225054339025845809877 6e-0012L,
154    3.32216956159789000423148243123645275262 4e-0015L,
155    2.82121329531400092425222648630572680509 3e-0018L,
156 };

158 GENERIC
159 y0l(x) GENERIC x;{
160         GENERIC z, d, s, c, ss, cc, u, v;
161         int i;

163         if(isnanl(x)) return x+x;
164         if(x <= zero){
165                 if(x==zero)
166                     d= -one/(x-x);
167                 else
168                     d = zero/(x-x);
169         }
170         if(x > 1.28L){
171                 if(!finitel(x)) return zero;
172                 s = sinl(x);
173                 c = cosl(x);
174         /* j0(x) = sqrt(2/(pi*x))*(p0(x)*cos(x0)-q0(x)*sin(x0))
175          * where x0 = x-pi/4
176          *      Better formula:
177          *              cos(x0) = cos(x)cos(pi/4)+sin(x)sin(pi/4)
178          *                      = 1/sqrt(2) * (cos(x) + sin(x))
179          *              sin(x0) = sin(x)cos(pi/4)-cos(x)sin(pi/4)
180          *                      = 1/sqrt(2) * (sin(x) - cos(x))
181          * To avoid cancellation, use
182          *              sin(x) +- cos(x) = -cos(2x)/(sin(x) -+ cos(x))
183          * to compute the worse one.
184          */
185                 if(x>1.0e2450L) {       /* x+x may overflow */
186                         ss = s-c;
187                         cc = s+c;
188                 } else if(signbitl(s)!=signbitl(c)) {
189                         ss = s - c;
190                         cc = -cosl(x+x)/ss;
191                 } else {
192                         cc = s + c;
193                         ss = -cosl(x+x)/cc;
194                 }
```



```
195         /*
196          * j0(x) = 1/sqrt(pi*x) * (P(0,x)*cc - Q(0,x)*ss)
197          * y0(x) = 1/sqrt(pi*x) * (P(0,x)*ss + Q(0,x)*cc)
198          */
199                 if(x>1.0e120L) return (invsqrtpi*ss)/sqrtl(x);
200                 return invsqrtpi*(pzero(x)*ss+qzero(x)*cc)/sqrtl(x);

202         }
203         if(x<=tiny) {
204             return(u0[0] + tpi*logl(x));
205         }
206         z = x*x;
207         u = u0[7]; v = v0[7];
208         for(i=6;i>=0;i--){
209             u = u*z + u0[i];
210             v = v*z + v0[i];
211         }
212         return(u/v + tpi*(j0l(x)*logl(x)));
213 }

215 static GENERIC pr0[12] = {      /* [16 -- inf] */
216    9.99999999999999999999999999999999997515e-0001L,
217    1.06598161537727337642536582396755059835 8e-0003L,
218    4.39099120092758897830637471898424071913 0e-0005L,
219    9.07208621860798671184706940733932136310 3e-0007L,
220    1.02255288617737536740840850104646167152 8e-0010L,
221    6.42076691224365824157063585408959726903 1e-0011L,
222    2.20645172512693391359108021108124226690 8e-0013L,
223    3.92836959681689507736370547874334629836 8e-0014L,
224    3.25815992887412459728670111972148287659 6e-0015L,
225    1.02571580813418897886067913014068510134 8e-0016L,
226    7.53717087479572125579600168702403128068 5e-0015L,
227   -1.57941390145015733230774558600420768779 6e-0014L,
228 };
229 static GENERIC ps0[11] = {
230    1.0e0L,
231    1.06605192787727337642536582396755051268 7e-0003L,
232    4.39173964716838159239917380432926635303 8e-0005L,
233    9.07516226180134367180565829412388886788 4e-0007L,
234    1.02318611851990475181958191207598599505 8e-0010L,
235    6.42786186041422374634051537651273027506 1e-0011L,
236    2.21086150323782358973548130362799340623 5e-0013L,
237    3.94324733578429290591595684090181817798 9e-0014L,
238    3.28372097677754514215020011064727000448 1e-0015L,
239    1.04534691881275404890364564153872898675 9e-0016L,
240    8.04345546806561890075059958429119368046 3e-0015L,
241 };
242 static GENERIC pr1[12] = {      /* [8 -- 16] */
243    9.99999999999999999999978442270110868361 8e-0001L,
244    6.79609853294833420775548869277790706289 4e-0002L,
245    1.84003611260572216882453075879716983604 2e-0005L,
246    2.59849048319191663726489434063584759812 2e-0007L,
247    2.10577486324270702552573024947205457852 3e-0009L,
248    1.01582204423054242666631499779694497995 9e-0011L,
249    2.93155745700811043676407769994418907187 5e-0012L,
250    4.96288512112545763365525922417932280882 4e-0013L,
251    4.70542405514822326915543059856335156627 9e-0014L,
252    2.29443985491074722915205608091042700111 0e-0015L,
253    4.90553184313748669150095001932247545862 9e-0015L,
254    3.18754316971033921879344254284573599456 5e-0015L,
255 };
256 static GENERIC ps1[14] = {
257    1.0e0L,
258    6.79680165794833420775457157606675818028 8e-0002L,
259    1.84051289120130056732542105982667636644 7e-0005L,
260    2.59977702831291897530625216712769507522 1e-0007L,
```

```
   2.10758257277104763684681128463424 4892537e+0009L,
   1.01727579469415610897578276388979940348e+0011L,
   2.93848764519246384542805975545476231601 1e+0012L,
   4.98251216473555705452104291618231792446 6e+0013L,
   4.73763990015370327479267746826456436143 7e+0014L,
   2.32339871912374274352424952827509710064 6e+0015L,
   5.03341910706921057786890979789698443919 1e+0015L,
   3.40903610593106860960131707675980471605 9e+0015L,
   7.50565536435267973758574514775352166216 6e+0013L,
  -9.97683715398368825078019824829710911831 3e+0012L,
};
static GENERIC pr2[12] = {      /* [5 -- 8 ] */
   9.99999999999999993785723678927736632022 0e-0001L,
   3.69284876526864957165160242037635884921 4e+0002L,
   5.37302206753547657692671590057760985410 e+0004L,
   4.03873889119131496997150403505721943072 5e+0006L,
   1.72828570630694052339738556665976264699 9e+0008L,
   4.37540081964588991115868873720605478853 4e+0009L,
   6.59895041820491240837559121778208856707 6e+0010L,
   5.82718203918323849248027540152007279378 3e+0011L,
   2.88422264291349239088757241499949097584 4e+0012L,
   7.37327887379776772193283783062868863277 5e+0012L,
   8.33829545756897376120507796439796923048 9e+0012L,
   2.91138318346728834577230881720980692214 3e+0012L,
};
static GENERIC ps2[14] = {
   1.0e0L,
   3.69355189026864947728889626717199321310 2e+0002L,
   5.37560788099836150247471513382806851429 7e+0004L,
   4.04247776402410824974499886257278636732 8e+0006L,
   1.73106983873701695668583958867013293951 3e+0008L,
   4.38714767404989877873822658593549141772 8e+0009L,
   6.62805865962065376534955694056771525816 5e+0010L,
   5.86965990416417774047168585636732216066 4e+0011L,
   2.91983944562281701705897755963896943638 3e+0012L,
   7.53531489769667140262820371861230925390 7e+0012L,
   8.69635556145293377577330985974861065893 5e+0012L,
   3.21615510314153722117360155769708321625 7e+0012L,
   4.75685708106894224824688015921378908636 3e+0010L,
  -3.49635661966660803223107486648147284067 e+0009L,
};
static GENERIC pr3[13] = {      /* [3.5 -- 5 ] */
   9.99999999999991669310728561239819658882 47e-0001L,
   2.26397592128291772119442532048497433694 5e+0002L,
   1.99435838674424584888949276278148419996 6e+0004L,
   8.98006745843054224355996249383166132316 8e+0005L,
   2.28221378752137266370556775642008755350 8e+0007L,
   3.40978437488906361825028869990837513592 3e+0008L,
   3.02438085740144858925434351758981171110 8e+0009L,
   1.57111036804674024689507172144308228637 9e+0010L,
   4.60318702024360463215368530046316059376 8e+0010L,
   7.08719645340971271944954928066405879340 3e+0010L,
   5.04619602177634635680368740964423906504 1e+0010L,
   1.28775843908016576570915427661885479993 2e+0010L,
   5.90067977341502343378784665809681359078 4e+0008L,
};
static GENERIC ps3[13] = {
   1.0e0L,
   2.26467904628285506132860461923177474711 6e+0002L,
   1.99593952398894455375565325538981210344 8e+0004L,
   8.99385314470634872703838996749018323682 0e+0005L,
   2.28832609963458884390698998370479546877 3e+0007L,
   3.42496710025524088516924095680479011828 2e+0008L,
   3.04631179797246399136802375964002891001 6e+0009L,
   1.58961496193282681279022479700797224003 e+0010L,
   4.69240662452774481649708913932507393992 7e+0010L,
```



```
   7.32048649590200891286646284907310832394 8e+0010L,
   5.34594597282897828993530959774298136099 4e+0010L,
   1.44403309191042375412130991509224717100 8e+0010L,
   7.98771468511531466837895727382438361052 5e+0008L,
};
static GENERIC pr4[13] = {      /* [2.5 , 3.5] */
   9.99999999998673667796111872274775712260 e-0001L,
   1.45382498070380055903787312356837884566 3e+0002L,
   8.09732721643068228826761044700650866103 2e+0003L,
   2.27384725203826437023116968638019266213 5e+0005L,
   3.56105672804621111135475999897698544962 2e+0006L,
   3.24493358880009637843462702936968037859 9e+0007L,
   1.74011239286071795037621003890847679258 8e+0008L,
   5.42617018745589328519787856388157926952 4e+0008L,
   9.49010748645436232100437733602052628137 1e+0008L,
   8.68887243942847004980171412107000531380 6e+0008L,
   3.67331585316643722281191065690012321551 5e+0008L,
   5.57777047035930330516487744633969327023 9e+0007L,
   1.54043864203168964130819788018129186571 4e+0006L,
};
static GENERIC ps4[13] = {      /* [2.5 , 3.5] */
   1.0e0L,
   1.45452810569815943977303595195913179981 6e+0002L,
   8.10744221520039239717217990043498785961 8e+0003L,
   2.27939039377824288757417709660328994140 e+0005L,
   3.57625162559225200842478111177093413584 4e+0006L,
   3.26790949905693263140594205867093381386 3e+0007L,
   1.76002151533080553749977823809970464880 5e+0008L,
   5.52555378766735398124206022258746572672 9e+0008L,
   9.76987029591282045788938408267126932851 1e+0008L,
   9.11058207100477427922690562962401800845 4e+0008L,
   3.98185767862195559937196768034391845434 5e+0008L,
   6.48240468623076939907319296166769703670 6e+0007L,
   2.21004694309587840244353546032939178229 8e+0006L,
};
static GENERIC pr5[13] = {      /* [1.777..., 2.5] */
   9.99999999911498610795181787114465588069 9e-0001L,
   9.25258373604858834256834457031543594761 4e+0001L,
   3.21872675785607871521463150240738626463 7e+0003L,
   5.55400996462111165647958850586257704083 1e+0004L,
   5.26999311564366433825319694452351029017 5e+0005L,
   2.87461377377843069119291219061822054457 5e+0006L,
   9.13353815110365835387414691961344243603 5e+0006L,
   1.67306704141033892282519301307735424919 3e+0007L,
   1.70691387384839801174479028920015184049 8e+0007L,
   9.06776658385328853455160023557674718679 e+0006L,
   2.21674673345788456853269535503633865587 2e+0006L,
   1.94575388080287254123570381272234451440 5e+0005L,
   3.13237441292194807153919563888533095174 9e+0003L,
};
static GENERIC ps5[13] = {      /* [1.777..., 2.5] */
   1.0e0L,
   9.25961498386218111888383167099034005298 2e+0001L,
   3.22512527546290338484212407513260929030 4e+0003L,
   5.57570536282910154529276005594185524649 2e+0004L,
   5.30604986303708785549617012195844849252 2e+0005L,
   2.90706075887350956430972990310901859721 5e+0006L,
   9.29805920658499589829825782713120853928 9e+0006L,
   1.72039107100696317683610802655654706298 0e+0007L,
   1.78261481292286519047939450948794192061 2e+0007L,
   9.70801638960527315353645203283987995015 5e+0006L,
   2.47649508468817009648021564096217514002 7e+0006L,
   2.36320066036558575966807779019460491718 7e+0005L,
   4.80323956984819607712120357570435693673 1e+0003L,
};
static GENERIC pr6[13] = {      /* [1.28, 1.777...] */
```

```
393    9.99999996977709549599860692552432255955 6e-0001L,
394    5.82548671946619443050328382409687221921 6e+0001L,
395    1.24815549163775728191518482496537990538 0e+0003L,
396    1.30209319984235860932133841707171047761 5e+0004L,
397    7.35383580418629278284096199981054301603 9e+0004L,
398    2.35647166111368618054919509255575134175 7e+0005L,
399    4.35055326742900958163298706094278084710 1e+0005L,
400    4.58876266187660063871915982665238941823 5e+0005L,
401    2.67579639854852343654422104522529012861 1e+0005L,
402    8.07764955710897138829829291998844994404 64e+0004L,
403    1.11764045922130687351906874166405457377 6e+0004L,
404    5.54440007239681469517578751155775788558 5e+0002L,
405    5.07255054119148049843128908990582291071 8e+0000L,
406 };
407 static GENERIC ps6[13] = {      /* [1.28, 1.777...] */
408    1.0e0L,
409    5.83251792535716505063907584818361306329 1e+0001L,
410    1.25214436474359212817125610436497646689 8e+0003L,
411    1.31030023434221681357911802241558574077 2e+0004L,
412    7.43466769709381219781729215403286363292 3e+0004L,
413    2.39870659558771916572646900240400461471 1e+0005L,
414    4.47273517625103157004869372427480602511 e+0005L,
415    4.78631352333776197529417142906703772361 1e+0005L,
416    2.85116187287273122847253606186536537019 2e+0005L,
417    8.89164826989914841233191802180138581558 6e+0004L,
418    1.29709748953535151757297812358475104228 7e+0004L,
419    7.09676164054597575620218414340046981261 8e+0002L,
420    8.37804933859023332597770240173334082035 1e+0000L,
421 };
422 static GENERIC sixteen = 16.0L;
423 static GENERIC huge    = 1.0e30L;

425 static GENERIC pzero(x)
426 GENERIC x;
427 {
428         GENERIC s,r,t,z;
429         int i;
430         if(x>huge) return one;
431         t = one/x; z = t*t;
432         if(x>sixteen) {
433             r = z*pr0[11]+pr0[10]; s = ps0[10];
434             for(i=9;i>=0;i--) {
435                 r = z*r + pr0[i];
436                 s = z*s + ps0[i];
437             }
438         } else if (x > eight){
439             r = pr1[11]; s = ps1[11]+z*(ps1[12]+z*ps1[13]);
440             for(i=10;i>=0;i--) {
441                 r = z*r + pr1[i];
442                 s = z*s + ps1[i];
443             }
444         } else if (x > five){   /* x > 5.0 */
445             r = pr2[11]; s = ps2[11]+z*(ps2[12]+z*ps2[13]);
446             for(i=10;i>=0;i--) {
447                 r = z*r + pr2[i];
448                 s = z*s + ps2[i];
449             }
450         } else if( x>3.5L) {
451             r = pr3[12]; s = ps3[12];
452             for(i=11;i>=0;i--) {
453                 r = z*r + pr3[i];
454                 s = z*s + ps3[i];
455             }
456         } else if( x>2.5L) {
457             r = pr4[12]; s = ps4[12];
458             for(i=11;i>=0;i--) {
```



```
459                 r = z*r + pr4[i];
460                 s = z*s + ps4[i];
461             }
462         } else if( x> (1.0L/0.5625L)){
463             r = pr5[12]; s = ps5[12];
464             for(i=11;i>=0;i--) {
465                 r = z*r + pr5[i];
466                 s = z*s + ps5[i];
467             }
468         } else {        /* assume x > 1.28 */
469             r = pr6[12]; s = ps6[12];
470             for(i=11;i>=0;i--) {
471                 r = z*r + pr6[i];
472                 s = z*s + ps6[i];
473             }
474         }
475         return r/s;
476 }
477

479 static GENERIC qr0[12] = {      /* [16, inf] */
480   -1.24999999999999999999999999999972972e-0001L,
481   -1.42517959554567057741439576250399159689 7e+0002L,
482   -6.31249964562597084553446025793622240721 9e+0004L,
483   -1.41137432645720838431512124369881446848 e+0007L,
484   -1.73503421275887358141098475786078725284 2e+0009L,
485   -1.19977764751278948942182634248505528068 0e+0011L,
486   -4.59602533408165571449986040969910037364 4e+0012L,
487   -9.26252562820128410779292447703165339918 7e+0013L,
488   -8.85839472868503924534439884218066286763 9e+0014L,
489   -3.26752795368753488762374062270950597211 3e+0015L,
490   -2.66422297118631196758712934702945006201 9e+0015L,
491    3.44246406072398786958518009534450410020 4e+0014L,
492 };
493 static GENERIC qs0[11] = {
494    1.0e0L,
495    1.14072961393653646193151661000318568788 1e+0003L,
496    5.05666551044229935100919818649008580358 0e+0005L,
497    1.13204176382564278794394165052271819911 5e+0008L,
498    1.39457011187258160639262067821424647976 7e+0010L,
499    9.67794521815226478953443107956374437842 1e+0011L,
500    3.73114032785153682822514305889634850209 6e+0013L,
501    7.61278595106486929172284668102088167641 0e+0014L,
502    7.47607701640676489173019100481186397594 0e+0015L,
503    2.95124648261359203542150342710039383170 9e+0016L,
504    3.10836180369181171113685458707430203490 1e+0016L,
505 };
506 static GENERIC qr1[12] = {      /* [8, 16 ] */
507   -1.24999999999999999999979490103834338181 57e-0001L,
508   -9.05121516639382264063675224489512412693 4e+0001L,
509   -2.62078270342814883767117903190420830394 7e+0004L,
510   -3.97557126155350445776617797450878579088 4e+0006L,
511   -3.47902933075931130627007221807407499409 0e+0008L,
512   -1.82395500812426857303621674618623982908 9e+0010L,
513   -5.76593269711180137576515602922156866443 5e+0011L,
514   -1.07984368079874259295400219241793477911 4e+0013L,
515   -1.14689363050459273908220576461158133289 7e+0014L,
516   -6.36701605968389846493610447282880704182 e+0014L,
517   -1.58310904196121349046445911190348420909 8e+0015L,
518   -1.23014955576424247310312865013579563941 2e+0015L,
519 };
520 static GENERIC qs1[14] = {
521    1.0e0L,
522    7.24683150811505811243857984777801445843 2e+0002L,
523    2.10085418443916851839938786306927037611 e+0005L,
524    3.19263641883795150743018828594099423512 2e+0007L,
```

```
525    2.80155844338335467453844346112443421 6152e+0009L,
526    1.47502699766437373929348392725065346748 7e+0011L,
527    4.69448682491395460855236382179992714531 8e+0012L,
528    8.89035010091920025083843870960154733402 1e+0013L,
529    9.62684442908290514487470106876046975206 7e+0014L,
530    5.54111074460046077352826386268752164214 0e+0015L,
531    1.48650049478945255672747032932321230965 63e+0016L,
532    1.41584010484595940036543077373209389921 0e+0016L,
533    1.78086609524151741808131256723968233648 3e+0015L,
534   -2.35923091738488935788763154407990129494 e+0014L,
535 };
536 static GENERIC qr2[12] = {      /* [5, 8] */
537   -1.24999999999999953193774436252777218164 e-0001L,
538   -4.94437389735696977483937597723924157396 6e+0001L,
539   -7.72844917543346528531426165007845047390 9e+0003L,
540   -6.26257432961275234633690143465122070590 3e+0005L,
541   -2.90094822022094330602723521742438067273 2e+0007L,
542   -7.98871964763419277046391715756287411953 5e+0008L,
543   -1.31822817192718138954776002666265701237 5e+0010L,
544   -1.28243977398302924530926327194542492819 6e+0011L,
545   -7.05092557082781804018614994025791884513 8e+0011L,
546   -2.02175188257387199000420561687420268442 9e+0012L,
547   -2.59293996240066855238433390057381263565 8e+0012L,
548   -1.03826710951889126284060151493297285032 6e+0012L,
549 };
550 static GENERIC qs2[14] = {
551    1.0e0L,
552    3.96135849288557000320278402289424895211 6e+0002L,
553    6.20578873886470188282875263458651092696 8e+0004L,
554    5.04571560393267028655067381301176440674 9e+0006L,
555    2.34924861136265832335334338943096875142 9e+0008L,
556    6.52024452441582863591768355372188006391 1e+0009L,
557    1.08911121122350771933706715988628188772 2e+0011L,
558    1.08040600090535986795877940941490301861 0e+0012L,
559    6.13564528089551470351415468062376956214 8e+0012L,
560    1.86243304024662587424586715136864366821 5e+0013L,
561    2.66778080578664888884077788702193708994 e+0013L,
562    1.39440110728908777476530071180931311282 4e+0013L,
563    1.09324750061632037556289829715672244548 4e+0012L,
564   -7.22887553037892872282660421649149378077 5e+0010L,
565 };
566 static GENERIC qr3[13] = {      /* [3.5 5] */
567   -1.24999999999947306774842037957848166107 5e-0001L,
568   -3.04454904863528935191357432480325097799 8e+0001L,
569   -2.89008114064976907849669300352468144086 9e+0003L,
570   -1.40492245681720223587934327533052910768 4e+0005L,
571   -3.86274661438557344351817740361734928186 9e+0006L,
572   -6.25751730911024904920113370891115504768 9e+0007L,
573   -6.03145133092083991698707978272732347752 0e+0008L,
574   -3.41154240517383061145402576575585438234 6e+0009L,
575   -1.08939247814972667213301449872302152609 9e+0010L,
576   -1.82493407842021094129014090341595678272 6e+0010L,
577   -1.40078027830435871042348107048693953113 9e+0010L,
578   -3.71648413606491736392663571674377109209 3e+0009L,
579   -1.39759107529642552997043489095490433158 0e+0008L,
580 };
581 static GENERIC qs3[13] = {
582    1.0e0L,
583    2.44149861390496204939100018701494585804 2e+0002L,
584    2.32618888207237071150016422234151433704 3e+0004L,
585    1.13713821312123133849497710465923957816 5e+0006L,
586    3.15291807073566272872299845260536425351 7e+0007L,
587    5.17287799342650725931427048844401359510 8e+0008L,
588    5.08308643973166980745596107885647077411 5e+0009L,
589    2.96184273206643412311932552113947690994 1e+0010L,
590    9.91218586686244073582978185608135315139 0e+0010L,
```



```
591    1.79356056125162223443056418156729798359 8e+0011L,
592    1.57709011934122812252526510849794040307 3e+0011L,
593    5.50991030678016619433388999998546368163 6e+0010L,
594    4.76169113407887449120232018151793675814 1e+0009L,
595 };
596 static GENERIC qr4[13] = {      /* [2.5 3.5] */
597   -1.24999999992856773433974504349070534083 5e-0001L,
598   -1.96720174873141906305160162443556552848 1e+0001L,
599   -1.18632914671456223640709974061552817070 7e+0003L,
600   -3.60773695922294181035630149115245793406 0e+0004L,
601   -6.11920071797810490493282846857519426712 5e+0005L,
602   -6.03784781158358226670305078652205586384 e+0006L,
603   -3.50355815333614035970053672039356598474 0e+0007L,
604   -1.18019647826822571875721852374678730977 3e+0008L,
605   -2.22186023208513491584142636350516968052 8e+0008L,
606   -2.17337250545274758529617676170174623676 0e+0008L,
607   -9.64936486506123755851773053950656801396 3e+0007L,
608   -1.46542922784793303454603964009486265038 5e+0007L,
609   -3.08300319792026208517058186624666338060 7e+0005L,
610 };
611 static GENERIC qs4[13] = {      /* [2.5 3.5] */
612    1.0e0L,
613    1.57962077373225914275261414213998685405 5e+0002L,
614    9.58137222032913873320387950375368505496 8e+0003L,
615    2.93959867237910809577611413101082588530 8e+0005L,
616    5.05218304931454221863034181869258844816 8e+0006L,
617    5.08349769559520663943383932633897198014 9e+0007L,
618    3.03638536180055338804971901400509920651 6e+0008L,
619    1.06782648145275340991056378516166149213 7e+0009L,
620    2.14564412555711804472074177512531966927 2e+0009L,
621    2.32411561595971994936394667349155221679 9e+0009L,
622    1.22326296211207075796695985561984701114 6e+0009L,
623    2.56976555331849542373847858594711027070 9e+0008L,
624    1.35474474429922712789790578773236565504 e+0007L,
625 };
626 static GENERIC qr5[13] = {      /* [1.777.., 2.5] */
627   -1.24999999593663969763768042817457606997 1e-0001L,
628   -1.26084605537131145348589192342648906831 5e+0001L,
629   -4.77239846754446748080117433029014157889 5e+0002L,
630   -8.93985259999029848661376083399649059972 4e+0003L,
631   -9.18407078714954205097954222644613424319 7e+0004L,
632   -5.40603894501827445836263789773928043517 1e+0005L,
633   -1.84589654470519026101865372867817108441 8e+0006L,
634   -3.61361699068080950187866757065330807154 7e+0006L,
635   -3.90878297813569325225255772041434862377 9e+0006L,
636   -2.17371102251732392710913817058844276817 6e+0006L,
637   -5.43125313067991848583640854900785624449 5e+0005L,
638   -4.59109854645268451008259158727594076595 9e+0004L,
639   -5.24471136416820780683552005779222964657 8e+0002L,
640 };
641 static GENERIC qs5[13] = {      /* [1.777.., 2.5] */
642    1.0e0L,
643    1.01453621085129087835089275097247486144 7e+0002L,
644    3.87554751068713531406443416009613968107 6e+0003L,
645    7.36191312267007981495525928199561773258 0e+0004L,
646    7.72028894421877112658108653958529314636 e+0005L,
647    4.68152955444675249640443143360830655803 8e+0006L,
648    1.66788262194050392545503125230836774582 0e+0007L,
649    3.46940315376139988188827262085530515624 1e+0007L,
650    4.09699204796421071186708938471994786301 9e+0007L,
651    2.59680475582921744931153073595956063055 4e+0007L,
652    7.98393377469788923815446506401941076384 5e+0006L,
653    9.81813381697990081908724242528075793815 2e+0005L,
654    3.06108393086869439601354153567074544356 0e+0004L,
655 };
```

```
static GENERIC qr6[13] = {      /* [1.28, 1.777..] */
  -1.24999988157728900180713728282492908 2771e-0001L,
  -7.99827351005311075961081059411953361 9282e+0000L,
  -1.87248195533517254336908961777156563 2719e+0002L,
  -2.12211678672630080507987400330379964 6812e+0003L,
  -1.29385028583952928250317826348477347 8457e+0004L,
  -4.44502474226631618103335419226252935 6093e+0004L,
  -8.73016137833435776766834446735650534 7070e+0004L,
  -9.70622289517207844280144497250531505 4736e+0004L,
  -5.89632551825985827016553151361819532 1041e+0004L,
  -1.82317203436810882227642082707466883 2233e+0004L,
  -2.50930417863505592663883304033747238 7175e+0003L,
  -1.15660896571577923731676982894172996 4099e+0002L,
  -7.02800578965073139688734682639778521 0442e-0001L,
};
static GENERIC qs6[13] = {      /* [1.28, 1.777..] */
    1.0e0L,
    6.45721108505806484560126132127772107 5900e+0001L,
    1.53400521658801121034282455513600868 2950e+0003L,
    1.77721799917644178259335766046237909 7171e+0004L,
    1.11837265264246946809108481026323119 9696e+0005L,
    4.01524243385846181314236574838647360 5294e+0005L,
    8.37708104551709864544861651438828049 7673e+0005L,
    1.01149502000801035257539800960416428 7337e+0006L,
    6.88672207529043056865222787520020895 5970e+0005L,
    2.50473518994802147204715714861317195 6537e+0005L,
    4.40813892017104484694100184435200981 7062e+0004L,
    3.10557217807211514567305872285364085 4884e+0003L,
    5.58829482111891611343739650457381703 3678e+0001L,
};
static GENERIC qzero(x)
GENERIC x;
{
        GENERIC s,r,t,z;
        int i;
        if(x>huge) return -0.125L/x;
        t = one/x; z = t*t;
        if(x>sixteen) {
            r = z*qr0[11]+qr0[10]; s = qs0[10];
            for(i=9;i>=0;i--) {
                r = z*r + qr0[i];
                s = z*s + qs0[i];
            }
        } else if(x>eight) {
            r = qr1[11]; s = qs1[11]+z*(qs1[12]+z*qs1[13]);
            for(i=10;i>=0;i--) {
                r = z*r + qr1[i];
                s = z*s + qs1[i];
            }
        } else if(x>five){  /* assume x > 5.0 */
            r = qr2[11]; s = qs2[11]+z*(qs2[12]+z*qs2[13]);
            for(i=10;i>=0;i--) {
                r = z*r + qr2[i];
                s = z*s + qs2[i];
            }
        } else if(x>3.5L) {
            r = qr3[12]; s = qs3[12];
            for(i=11;i>=0;i--) {
                r = z*r + qr3[i];
                s = z*s + qs3[i];
            }
        } else if(x>2.5L) {
            r = qr4[12]; s = qs4[12];
            for(i=11;i>=0;i--) {
                r = z*r + qr4[i];
                s = z*s + qs4[i];
```



```
            }
        } else if(x> (1.0L/0.5625L)) {
            r = qr5[12]; s = qs5[12];
            for(i=11;i>=0;i--) {
                r = z*r + qr5[i];
                s = z*s + qs5[i];
            }
        } else {        /* assume x > 1.28 */
            r = qr6[12]; s = qs6[12];
            for(i=11;i>=0;i--) {
                r = z*r + qr6[i];
                s = z*s + qs6[i];
            }
        }
        return t*(r/s);
}
```

```
**********************************************************
   28046 Sun May  4 03:05:28 2014
new/usr/src/lib/libm/common/LD/j1l.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.
 */

/*
 * floating point Bessel's function of the first and second kinds
 * of order zero: j1(x),y1(x);
 *
 * Special cases:
 *      y0(0)=y1(0)=yn(n,0) = -inf with division by zero signal;
 *      y0(-ve)=y1(-ve)=yn(n,-ve) are NaN with invalid signal.
 */

#pragma weak j1l = __j1l
#pragma weak y1l = __y1l

#include "libm.h"
#include "libm_synonyms.h"
#include "longdouble.h"
#endif /* ! codereview */
#include <math.h>
#if defined(__SUNPRO_C)
#include <sunmath.h>
#endif

#define GENERIC long double
static GENERIC
zero    = 0.0L,
small   = 1.0e-9L,
tiny    = 1.0e-38L,
one     = 1.0L,
five    = 5.0L,
invsqrtpi= 5.641895835477562869480794515607725858441e-0001L,
tpi     = 0.636619772367581343075535053490057448L;

static GENERIC pone(), qone();
static GENERIC r0[7] = {
```



```
  -6.249999999999999999999999999999627320e-0002L,
   1.940606727194041716205384618494641565464e-0003L,
  -3.005630423155733701856481469986459043883e-0005L,
   2.345586219403918667468341047369572169358e-0007L,
  -9.976809285885253587529010109133336669724e-0010L,
   2.218743258363623946078958783775107473381e-0012L,
  -2.071079656218700604767650924103578046280e-0015L,
};
static GENERIC s0[7] = {
   1.0e0L,
   1.061695903156199920738051277075003059555e-0002L,
   5.521860513111180371566951179398862692060e-0005L,
   1.824214367413754193524107877084979441407e-0007L,
   4.098957778439576834818838198039029353925e-0010L,
   6.047735079699663898532400909252640561976-0013L,
   4.679044728878836197247923279512047035041e-0016L,
};

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

        if(!finitel(x)) return one/x;
        sgn = signbitl(x);
        x = fabsl(x);
        if(x > 1.28L){
                s = sinl(x);
                c = cosl(x);
        /* j1(x) = sqrt(2/(pi*x))*(p1(x)*cos(x0)-q1(x)*sin(x0))
         * where x0 = x-3pi/4
         *      Better formula:
         *              cos(x0) = cos(x)cos(3pi/4)+sin(x)sin(3pi/4)
         *                      =  1/sqrt(2) * (sin(x) - cos(x))
         *              sin(x0) = sin(x)cos(3pi/4)-cos(x)sin(3pi/4)
         *                      = -1/sqrt(2) * (cos(x) + sin(x))
         * To avoid cancellation, use
         *              sin(x) +- cos(x) = -cos(2x)/(sin(x) -+ cos(x))
         * to compute the worse one.
         */
                if(x>1.0e2450L) {       /* x+x may overflow */
                        ss = -s-c;
                        cc =  s-c;
                } else if(signbitl(s)!=signbitl(c)) {
                        cc = s - c;
                        ss = cosl(x+x)/cc;
                } else {
                        ss = -s-c;
                        cc = cosl(x+x)/ss;
                }
        /*
         * j1(x) = 1/sqrt(pi*x) * (P(1,x)*cc - Q(1,x)*ss)
         * y1(x) = 1/sqrt(pi*x) * (P(1,x)*ss + Q(1,x)*cc)
         */
                if(x>1.0e120L) return (invsqrtpi*cc)/sqrtl(x);
                d =  invsqrtpi*(pone(x)*cc-qone(x)*ss)/sqrtl(x);
                if(sgn==0) return d; else return -d;
        }
        if(x<=small) {
            if(x<=tiny) d = 0.5L*x;
            else d =  x*(0.5L-x*x*0.125L);
            if(sgn==0) return d; else return -d;
        }
        z = x*x;
            r = r0[6];
            s = s0[6];
```

```
129             for(i=5;i>=0;i--) {
130                 r = r*z + r0[i];
131                 s = s*z + s0[i];
132             }
133         d = x*0.5L+x*(z*(r/s));
134         if(sgn==0) return d; else return -d;
135 }

137 static GENERIC u0[7] = {
138   -1.960570906462389484060557273467558703503e-0001L,
139    5.166389353148318460304315890665450006495e-0002L,
140   -2.229699464105910913337190798743451115604e-0003L,
141    3.625437034548863342715657067759078267158e-0005L,
142   -2.689902826993117212255524537353883987171e-0007L,
143    9.304570592456930912969387719010256018466e-0010L,
144   -1.234878126794286643318321347997500346131e-0012L,
145 };
146 static GENERIC v0[8] = {
147    1.0e0L,
148    1.369394302535807332517110204820556695644e-0002L,
149    9.508438148097659501433367062605935379588e-0005L,
150    4.399007309420092056052714797296467565655e-0007L,
151    1.488083087443756398305819693177715000787e-0009L,
152    3.751609832625793536245746965768587624922e-0012L,
153    6.680926434086257291872903276124244131448e-0015L,
154    6.676602383908906988160099057991121446058e-0018L,
155 };

157 GENERIC
158 y1l(x) GENERIC x;{
159         GENERIC z, s, c, ss, cc, u, v;
160         int i;

162         if(isnanl(x)) return x+x;
163         if(x <= zero){
164                 if(x==zero)
165                     return -one/zero;
166                 else
167                     return zero/zero;
168         }
169         if(x > 1.28L){
170                 if(!finitel(x)) return zero;
171                 s = sinl(x);
172                 c = cosl(x);
173         /* j1(x) = sqrt(2/(pi*x))*(p1(x)*cos(x0)-q1(x)*sin(x0))
174          * where x0 = x-3pi/4
175          *      Better formula:
176          *              cos(x0) = cos(x)cos(3pi/4)+sin(x)sin(3pi/4)
177          *                      =  1/sqrt(2) * (sin(x) - cos(x))
178          *              sin(x0) = sin(x)cos(3pi/4)-cos(x)sin(3pi/4)
179          *                      = -1/sqrt(2) * (cos(x) + sin(x))
180          * To avoid cancellation, use
181          *              sin(x) +- cos(x) = -cos(2x)/(sin(x) -+ cos(x))
182          * to compute the worse one.
183          */
184                 if(x>1.0e2450L) {       /* x+x may overflow */
185                         ss = -s-c;
186                         cc =  s-c;
187                 } else if(signbitl(s)!=signbitl(c)) {
188                         cc = s - c;
189                         ss = cosl(x+x)/cc;
190                 } else {
191                         ss = -s-c;
192                         cc = cosl(x+x)/ss;
193                 }
194         /*
```



```
195          * j1(x) = 1/sqrt(pi*x) * (P(1,x)*cc - Q(1,x)*ss)
196          * y1(x) = 1/sqrt(pi*x) * (P(1,x)*ss + Q(1,x)*cc)
197          */
198                 if(x>1.0e91L) return (invsqrtpi*ss)/sqrtl(x);
199                 return invsqrtpi*(pone(x)*ss+qone(x)*cc)/sqrtl(x);
200         }
201         if(x<=tiny) {
202             return(-tpi/x);
203         }
204         z = x*x;
205             u = u0[6]; v = v0[6]+z*v0[7];
206             for(i=5;i>=0;i--){
207                 u = u*z + u0[i];
208                 v = v*z + v0[i];
209             }
210         return(x*(u/v) + tpi*(j1l(x)*logl(x)-one/x));
211 }

213 static GENERIC pr0[12] = {
214    1.00000000000000000000000000000000000267e+0000L,
215    1.06071787504589145560218084327675800303 5e+0003L,
216    4.34434754289212702444668771218110585233 5e+0005L,
217    8.91568022072400701637792425271741045709 4e+0007L,
218    9.96950225993840606280987325756917127281 9e+0009L,
219    6.20029019313861303564651033870738631659 5e+0011L,
220    2.10597854878801511985181585442224733011 8e+0013L,
221    3.69663577278460123937173081031199836894 8e+0014L,
222    3.01591309792069468205795841253413451515 6e+0015L,
223    9.37029847133935309812327742732859272592 1e+0015L,
224    7.19034900519633596734079926507402944305 7e+0015L,
225    2.73609778624068999688039107492755251798 2e+0014L,
226 };
227 static GENERIC ps0[11] = {
228    1.0e0L,
229    1.06060068754589145560218084327675809510 7e+0003L,
230    4.34310609341697558914715390650533890096 1e+0005L,
231    8.91060586900217656658207224224435339905 9e+0007L,
232    9.95912205863508788869071391762205654019 0e+0009L,
233    6.18874496723494823179248294917104184389 4e+0011L,
234    2.09886397695378350640175987380199030490 7e+0013L,
235    3.67287035701806319674672975147993890845 0e+0014L,
236    2.97553841924682492104901152957438588842 0e+0015L,
237    9.06365765999504320501868602928447987370 1e+0015L,
238    6.40195334431474791672936644150889271169 1e+0015L,
239 };
240 static GENERIC pr1[12] = {
241    1.00000000000000000000002366752413066098 4e+0000L,
242    6.74615441997961875435480348812645297120 4e+0002L,
243    1.81121078108339015485701833029614597050 2e+0005L,
244    2.53309839037992426803800532909528784224 4e+0007L,
245    2.02968361980534214525233857087542460072 9e+0009L,
246    9.66085966219271146530106940159892998031 9e+0010L,
247    2.74339623864483151993409896771662131631 6e+0012L,
248    4.55309735414085437793102317026345524628 8e+0013L,
249    4.21024506985221975747616986497487072037 4e+0014L,
250    1.98733405622959648507664596717616980172 7e+0015L,
251    4.06712005278709689383897045575133893046 2e+0015L,
252    2.48653960638040639831084526491069139813 3e+0015L,
253 };
254 static GENERIC ps1[14] = {
255    1.0e0L,
256    6.74498254497961875435580868019685952178 2e+0002L,
257    1.81042179539696676203215529044136474035 0e+0005L,
258    2.53098646064431065152958375969998843557 3e+0007L,
259    2.02674327604802312136024928881829022414 5e+0009L,
260    9.63746192440740593524526940705264134183 6e+0010L,
```

```
   2.732378628423766417402292797028314160831e+0012L,
   4.522345274960527124354844364012184278488e+0013L,
   4.160650668341743132685335758415469856545e+0014L,
   1.943730242988858208243492424892435901211e+0015L,
   3.880228532692127989901131618598067450001e+0015L,
   2.178020816161154615841000173683302999728e+0015L,
  -8.994062666842225551554346698171600634173e+0013L,
   1.368520368508851253495764806934619574990e+0013L,
};
static GENERIC pr2[12] = {
   1.000000000000000006938651621840396237282e+0000L,
   3.658416291850404981407101077037948144698e+0002L,
   5.267073772170356547709794670602812447537e+0004L,
   3.912012101226837463014925210735894620442e+0006L,
   1.651295648974103957193874928714180765625e+0008L,
   4.114901144480797609972484998142146783499e+0009L,
   6.092524309766036681542980572526335147672e+0010L,
   5.263913178071282616719249969074134570577e+0011L,
   2.538408581124324223367341020538081330994e+0012L,
   6.288607929360291027895126983015365677648e+0012L,
   6.848330048211148419047055075386525945280e+0012L,
   2.290309646838867941423178163991423244690e+0012L,
};
static GENERIC ps2[14] = {
   1.0e0L,
   3.657244416850405086459410165762319861856e+0002L,
   5.262802358425023243992387075861237306312e+0004L,
   3.905896813959919648136295861661483848364e+0006L,
   1.646791907791461220742694842108202772763e+0008L,
   4.096132803064256022224954120208201437344e+0009L,
   6.046665195915950447544429445730680236759e+0010L,
   5.198061739781991313414052212328653295168e+0011L,
   2.484233851814333966401527626421254279796e+0012L,
   6.047868806925315879339651539434315255940e+0012L,
   6.333103831254091652501642567294101813354e+0012L,
   1.875143098754284994467609936924685024968e+0012L,
  -5.238330920563392692965412762508813601534e+0010L,
   4.656888609439364725427789198383779259957e+0009L,
};
static GENERIC pr3[13] = {
   1.000000000000009336887318068056137842897e+0000L,
   2.242719942728459588488051572002835729183e+0002L,
   1.955450611382026550266257737331095691092e+0004L,
   8.707143293993619899395400562409175590739e+0005L,
   2.186267894487004565948324289010954505316e+0007L,
   3.224328510541957792360691585667502864688e+0008L,
   2.821057355151380597331792896882741364897e+0009L,
   1.445371387295422404365584793796028979840e+0010L,
   4.181743160669891357783011002656658107864e+0010L,
   6.387371088767993119325536137794535513922e+0010L,
   4.575619999412716078064070587767416436396e+0010L,
   1.228415651211639160620284441690503550842e+0010L,
   7.242170349875563053436050532153112882072e+0008L,
};
static GENERIC ps3[13] = {
   1.0e0L,
   2.241548067728529551049804610486061401070e+0002L,
   1.952838216795552145132137932931237181307e+0004L,
   8.684574926493185744628127341069974575526e+0005L,
   2.176357771067037962940853412819852189164e+0007L,
   3.199958682356132977319258783167122100567e+0008L,
   2.786218931525334687844675219914201872570e+0009L,
   1.416283776951741549631417572317916039767e+0010L,
   4.042962659271567948735676834609348842922e+0010L,
   6.028168426466945100838472229684444021661e+0010L,
   4.118410226794641413833887606580085281111e+0010L,
```



```
   9.918735736297038430744161253338202230263e+0009L,
   4.092967198238098023219124487437130332038e+0008L,
};
static GENERIC pr4[13] = {
   1.000000000001509220978157399042059553390e+0000L,
   1.437551868378147851133499996323782607787e+0002L,
   7.911335537418177296041518061404505428004e+0003L,
   2.193710939115317214716518908935756104804e+0005L,
   3.390662495136730962513489796538274984382e+0006L,
   3.048655347929348891006070609293884274789e+0007L,
   1.613781633489496606354045161527450975195e+0008L,
   4.975089835037230277110156150038482159988e+0008L,
   8.636047087015115403880904418339566323264e+0008L,
   7.918202912328366140110671223076949101509e+0008L,
   3.423294665798984733439650311722794853294e+0008L,
   5.621904953441963961040503934782662613621e+0007L,
   2.086303543310240260758670404509484499793e+0006L,
};
static GENERIC ps4[13] = {
   1.0e0L,
   1.436379933845323716704933195918473623304e+0002L,
   7.894647154785430678061053848847436659499e+0003L,
   2.184659753392097529008981741550878586174e+0005L,
   3.366109083305465176803513738147049499361e+0006L,
   3.011911545968996817697665866587226343186e+0007L,
   1.582262913779689851316760148459414895301e+0008L,
   4.819268809494937919217938589530138201770e+0008L,
   8.201355762990450679702837123432527154830e+0008L,
   7.268232093982510937417446421282341425212e+0008L,
   2.950911909015572933262131323934036480462e+0008L,
   4.242839924305934423010858966540621219396e+0007L,
   1.064387620445090779182117666330405186866e+0006L,
};
static GENERIC pr5[13] = {
   1.000000000102434805241171427253847353861e+0000L,
   9.129332257083629259060502249025963234821e+0001L,
   3.132238483586953037576119377504557191413e+0003L,
   5.329782528269307971278943122454171107861e+0004L,
   4.988460157184117790692873002103052944145e+0005L,
   2.686620716157868161470103342560474693782e+0006L,
   8.445418526028961197703799808701268301831e+0006L,
   1.536575358646141157475725889907900827390e+0007L,
   1.568405818236523821796862770586544811945e+0007L,
   8.450876239888770102387618667362302173547e+0006L,
   2.154414900139567328424026827163203446077e+0006L,
   2.105656926565043898888460254808062352205e+0005L,
   4.739165011023396507022134303736862812975e+0003L,
};
static GENERIC ps5[13] = {
   1.0e0L,
   9.117613509595327476509152673394703847793e+0001L,
   3.121697972484015639301279229281770795147e+0003L,
   5.294447222735893568040911873834576440255e+0004L,
   4.930368882192772335798256684110887882807e+0005L,
   2.634854685641165298302167435798357437768e+0006L,
   8.185462775400326393555896157031818280918e+0006L,
   1.462417423080215192609668642663030667086e+0007L,
   1.450624993985851675982860844153954896015e+0007L,
   7.460467647561995283219086567162006113864e+0006L,
   1.754210981405612478869227142579056338965e+0006L,
   1.463286721155271971526264914524746699596e+0005L,
   2.155894725796702015341211116579827039459e+0003L,
};
static GENERIC pr6[13] = {
   1.000000003564855546741735920315743157129e+0000L,
   5.734003934862540458119423509909510288366e+0001L,
```

```
   1.20957249193585048608655969229179688797 6e+0003L,
   1.24339839142228124793367477916366028683 8e+0004L,
   6.93099675518143793725822099860170827878 7e+0004L,
   2.19806765953275759864672224996676762009 9e+0005L,
   4.03365943271205863393317911582057685845 5e+0005L,
   4.25775965721900802701604720657457435867 8e+0005L,
   2.51191739587600434948072127744576391638 9e+0005L,
   7.81375615307062365417873165138188195355 2e+0004L,
   1.15206917338112788138558809290586435289 1e+0004L,
   6.54858078280408855377781603755152339808 2e+0002L,
   8.66872537011690613232754276612793849688 0e+0000L,
};
static GENERIC ps6[13] = {
   1.0e0L,
   5.72228523635711456649922152573628620518 4e+0001L,
   1.20301084287831793544458295062033957050 6e+0003L,
   1.23005833578583550155825502172435371208 e+0004L,
   6.80099855060786128886530043864808989441 2e+0004L,
   2.13076782959930426298776934753685088592 1e+0005L,
   3.84048346664391668175993697299215531002 6e+0005L,
   3.94743237345922554286181914810808116039 3e+0005L,
   2.23781623939308111148158843445783852673 8e+0005L,
   6.54582049512441972339894627379092154077 4e+0004L,
   8.72956363032089274150072621327883473719 6e+0003L,
   4.13076266029189475345017479419699881370 9e+0002L,
   3.48036889867268464513033578601507559559 8e+0000L,
};
static GENERIC sixteen = 16.0L;
static GENERIC eight   = 8.0L;
static GENERIC huge    = 1.0e30L;

static GENERIC pone(x)
GENERIC x;
{
        GENERIC s,r,t,z;
        int i;
        if(x>huge) return one;
        t = one/x; z = t*t;
        if(x>sixteen) {
            r = z*pr0[11]+pr0[10]; s = ps0[10];
            for(i=9;i>=0;i--) {
                r = z*r + pr0[i];
                s = z*s + ps0[i];
            }
        } else if(x>eight) {
            r = pr1[11]; s = ps1[11]+z*(ps1[12]+z*ps1[13]);
            for(i=10;i>=0;i--) {
                r = z*r + pr1[i];
                s = z*s + ps1[i];
            }
        } else if(x>five) {
            r = pr2[11]; s = ps2[11]+z*(ps2[12]+z*ps2[13]);
            for(i=10;i>=0;i--) {
                r = z*r + pr2[i];
                s = z*s + ps2[i];
            }
        } else if( x>3.5L) {
            r = pr3[12]; s = ps3[12];
            for(i=11;i>=0;i--) {
                r = z*r + pr3[i];
                s = z*s + ps3[i];
            }
        } else if( x>2.5L) {
            r = pr4[12]; s = ps4[12];
            for(i=11;i>=0;i--) {
                r = z*r + pr4[i];
```



```
                s = z*s + ps4[i];
            }
        } else if( x> (1.0L/0.5625L)){
            r = pr5[12]; s = ps5[12];
            for(i=11;i>=0;i--) {
                r = z*r + pr5[i];
                s = z*s + ps5[i];
            }
        } else {        /* assume x > 1.28 */
            r = pr6[12]; s = ps6[12];
            for(i=11;i>=0;i--) {
                r = z*r + pr6[i];
                s = z*s + ps6[i];
            }
        }
        return r/s;
}

static GENERIC qr0[12] = {
   3.74999999999999999999999999999971033e-0001L,
   4.25672603523705060160768227743094262226e+0002L,
   1.87597649081287848919240997894540106606 6e+0005L,
   4.17031426804804191427360368031774559279 0e+0007L,
   5.09275013254385581729345111897455574655 1e+0009L,
   3.49474967627848865410350579579413948340 4e+0011L,
   1.32706214825743731699766781709669417370 9e+0013L,
   2.64899313827342722690750374206655115049 0e+0014L,
   2.51169566590954741222243049447399812768 4e+0015L,
   9.27469450666228904322431049916470230609 6e+0015L,
   8.15090417066366382933132030291179289200 2e+0015L,
  -5.00191873370766235577203782962038876512 2e+0014L,
};
static GENERIC qs0[11] = {
   1.0e0L,
   1.13540038022988016042871527398215576009 3e+0003L,
   5.00570118387712616432676554551659074436 0e+0005L,
   1.11344420011371216798433760393304010298 7e+0008L,
   1.36107481992522306277871756569903947112 4e+0010L,
   9.35575098580284948443893390532598280965 3e+0011L,
   3.56346278600898882500396554385799808482 8e+0013L,
   7.15514511390009416364872686380380291045 4e+0014L,
   6.87126683583447275805555901385184365411 3e+0015L,
   2.62203089922673671264497498815734523409 2e+0016L,
   2.60291272917287633065007702170613970774 6e+0016L,
};
static GENERIC qr1[12] = {
   3.74999999999999999999776245820728440580 6e-0001L,
   2.69788399888170683992925551749818998048 5e+0002L,
   7.75519592578102848938693887047383441101 9e+0004L,
   1.16677776210401777198211072895528968355 e+0007L,
   1.01150477298432116832001008452026106936 2e+0009L,
   5.24600770357415685357775457172020555001 0e+0010L,
   1.63769254988559268316611655169126653764 7e+0012L,
   3.02230362369818566991299031092503938249 5e+0013L,
   3.15476992729065568484610703026590998794 6e+0014L,
   1.71581991344155477008973093480812336092 1e+0015L,
   4.16504435575973262227353444513173618851 0e+0015L,
   3.15138142087417470564310038170808628759 6e+0015L,
};
static GENERIC qs1[14] = {
   1.0e0L,
   7.19709170535121823978563317240827698282 8e+0002L,
   2.07001279959954868554488304129760986105 5e+0005L,
   3.11701481531765622187184015277845875451 6e+0007L,
   2.70571967890255497486332587702590297172 7e+0009L,
```

```
   1.406113614727345726925060648750867264098e+0011L,
   4.403777536067131320363005978631674817359e+0012L,
   8.170725690209322283061499386703167242894e+0013L,
   8.609458844975495289227794126964431210566e+0014L,
   4.766766367015473481257280600694952920204e+0015L,
   1.202286587943342194863557940888115641650e+0016L,
   1.012474328306200909525063936061756024120e+0016L,
   6.183552022678917858273222879615824070703e+0014L,
  -9.756731548558226997573737400988225722740e+0013L,
};
static GENERIC qr2[12] = {
   3.749999999999994812456472622269942931 89e-0001L,
   1.471366807289771354491181140167359026735e+0002L,
   2.279432486768448220142080962843526951250e+0004L,
   1.828943048523771225163804043356958285893e+0006L,
   8.379828388647823135832220596417725010837e+0007L,
   2.279814029335044024585393671278378022053e+0009L,
   3.711653952257118120832817785271466441420e+0010L,
   3.557650914518554549916730572553105048068e+0011L,
   1.924583483146095896259774329498934160650e+0012L,
   5.424386256063736390759567088291887140278e+0012L,
   6.839325621241776786206509704671746841737e+0012L,
   2.702169563144001166291686452305436313971e+0012L,
};
static GENERIC qs2[14] = {
   1.0e0L,
   3.926379194439388135703211933895203191089e+0002L,
   6.089148804106598297488336063007609312276e+0004L,
   4.893546162973278583711376356041614150645e+0006L,
   2.247571119114497845046388801813832219404e+0008L,
   6.137635663350177751290469334200757872645e+0009L,
   1.005115019784102856424493519524998953678e+0011L,
   9.725664462014503832860151384604677240620e+0011L,
   5.345525100485511116148634192844434636072e+0012L,
   1.549944007398946691720862738173956994779e+0013L,
   2.067148441178952625710302124163264760362e+0013L,
   9.401565402641963611295119487242595462301e+0012L,
   3.548217088622398274748837287769709374385e+0011L,
  -2.934470341719047120076509938432417352365e+0010L,
};
static GENERIC qr3[13] = {
   3.749999999994127240845798332974514720 91e-0001L,
   9.058478580291706212422978492938435582527e+0001L,
   8.524056033161038750461083666711724381171e+0003L,
   4.105967158629109427753434569223631014730e+0005L,
   1.118326603378531348259783091972623333657e+0007L,
   1.794636683403578918528064904714132329343e+0008L,
   1.714314157463635959556133236004368896724e+0009L,
   9.622092032236084846572067257267661456030e+0009L,
   3.057759524485859159957762858780768355020e+0010L,
   5.129306780754798531609621454415938890020e+0010L,
   3.999122002794961070680636194346316041352e+0010L,
   1.122298454643493485989721564358100345388e+0010L,
   5.603981987645989709668830968522362582221e+0008L,
};
static GENERIC qs3[13] = {
   1.0e0L,
   2.418328663076578169836155170053634419922e+0002L,
   2.279620205900121042587523541281272875520e+0004L,
   1.100984222585729521470129014992217092794e+0006L,
   3.010743223679247091004262516286654516282e+0007L,
   4.860925542827367817289619265215599433996e+0008L,
   4.686668111035348691982715864307839581243e+0009L,
   2.668701788405102017427214705946730894074e+0010L,
   8.677395746106802640390580944836650584903e+0010L,
   1.511936455574951790658498795945106643036e+0011L,
```



```
   1.260845604432623478002018696873608353093e+0011L,
   4.052692278419853853911440231600864589805e+0010L,
   2.965516519212226064983267822433296947 29e+0009L,
};
static GENERIC qr4[13] = {
   3.749999999919234164154669754440123072618e-0001L,
   5.844218580776819864791168253485055101858e+0001L,
   3.489273514092912982675669411371435670220e+0003L,
   1.050523637774575684509663430018995479594e+0005L,
   1.764549172059701565500717319792780115289e+0006L,
   1.725532438844133795028063102681497371154e+0007L,
   9.938114847359778539965140247590176334874e+0007L,
   3.331710808184595545396883770200772842314e+0008L,
   6.271970557641881511609560444872797282698e+0008L,
   6.188529798677357075020774923903737913285e+0008L,
   2.821905302742849974509982167877885011629e+0008L,
   4.615467358646911976773290256984329814896e+0007L,
   1.348140608731546467396685802693380693275e+0006L,
};
static GENERIC qs4[13] = {
   1.0e0L,
   1.561192663112345185261418296389902133372e+0002L,
   9.346678031144098270547225423124213083072e+0003L,
   2.825851246482293547838023847601704751590e+0005L,
   4.776572711622156091710902891124911556293e+0006L,
   4.715106953717135402977938048006267859302e+0007L,
   2.753962350894311316439652227611209035193e+0008L,
   9.428501434615463207768964787500411575223e+0008L,
   1.832650858775206787088236896454141572617e+0009L,
   1.901697378939743226948920874296595242257e+0009L,
   9.433322226854293780627188599226380812725e+0008L,
   1.808520540608671608680284520798858587370e+0008L,
   7.983342331736662753157217446919462398008e+0006L,
};
static GENERIC qr5[13] = {
      3.749999955331364437028988850515190446719e-0001L,
      3.739356381766559882677514593041627547911e+0001L,
      1.399562500629413529355265462912819802551e+0003L,
      2.594154053098947925345332218062210111753e+0004L,
      2.640149879297408640394163979394594318371e+0005L,
      1.542471854873199142031889093591449397995e+0006L,
      5.242272868972053374067572098992335425895e+0006L,
      1.025834487769410221329633071426044839935e+0007L,
      1.116553924239448940142230579060124209622e+0007L,
      6.318076065595910176374916303525884653514e+0006L,
      1.641218086168640408527639735915512881785e+0006L,
      1.522369793529178644168813882912134706444e+0005L,
      2.526530541062297200914180060208669584055e+0003L,
};
static GENERIC qs5[13] = {
      1.0e0L,
      9.998960735935075380397545659016287506660e+0001L,
      3.758767417842043742686475060540416737562e+0003L,
      7.013652806952306520121959742519780781653e+0004L,
      7.208949808818615099246529616211730446850e+0005L,
      4.272753927109614455417836186072202009252e+0006L,
      1.482524411356470699336129814111025434703e+0007L,
      2.988750366665678233425279237627700803473e+0007L,
      3.396957890261080492694709150553619185065e+0007L,
      2.050652487738593004111578091156304540386e+0007L,
      5.900504120811732547616511555946279451316e+0006L,
      6.563391409260160897024498082273183468347e+0005L,
      1.692629845012790205348966731477187041419e+0004L,
};
static GENERIC qr6[13] = {
    3.749999861516664133157566870858975421296e-0001L,
```

```
    2.36786375674776486312079743159947346891 8e+0001L,
    5.47671580211497624888206732563079314377 7e+0002L,
    6.14319035786984289402501294544096170251 e+0003L,
    3.71625053467799785051373359514046385173 0e+0004L,
    1.27088346382387675213832690502287565743 0e+0005L,
    2.49530144963681448164637166542908380138 8e+0005L,
    2.78957898821295224834048629625439860194 2e+0005L,
    1.71824794691110905593181908713739732463 4e+0005L,
    5.45897321401166571433032673220410636422 9e+0004L,
    7.91210268668794878604894333975959665281 3e+0003L,
    4.07796100616086693572203071514908793809 1e+0002L,
    3.76520697277024508555105723788252851042 8e+0000L,
};
static GENERIC qs6[13] = {
    1.0e0L,
    6.34164653294051730564189385267392680960 1e+0001L,
    1.47705827741404079093259753792067102535 9e+0003L,
    1.67440656403104449143604425339353648760 4e+0004L,
    1.02851650136975594989505080690894650768 e+0005L,
    3.59362004253288529508746350773328543420 7e+0005L,
    7.26792499138102091518587339945372479962 5e+0005L,
    8.46227751076881839996119142620500608308 8e+0005L,
    5.51439989223089216337361189564550025051 4e+0005L,
    1.89808424100925935354062027293218810229 9e+0005L,
    3.10294124211773901572198412308102625306 8e+0004L,
    1.95897118443146690768144065018142108614 3e+0003L,
    2.87885335731049508718172161388945512186 7e+0001L,
};
static GENERIC qone(x)
GENERIC x;
{
        GENERIC s,r,t,z;
        int i;
        if(x>huge) return 0.375L/x;
        t = one/x; z = t*t;
        if(x>sixteen) {
            r = z*qr0[11]+qr0[10]; s = qs0[10];
            for(i=9;i>=0;i--) {
                r = z*r + qr0[i];
                s = z*s + qs0[i];
            }
        } else if(x>eight) {
            r = qr1[11]; s = qs1[11]+z*(qs1[12]+z*qs1[13]);
            for(i=10;i>=0;i--) {
                r = z*r + qr1[i];
                s = z*s + qs1[i];
            }
        } else if (x>five) {    /* x > 5.0 */
            r = qr2[11]; s = qs2[11]+z*(qs2[12]+z*qs2[13]);
            for(i=10;i>=0;i--) {
                r = z*r + qr2[i];
                s = z*s + qs2[i];
            }
        } else if(x>3.5L) {
            r = qr3[12]; s = qs3[12];
            for(i=11;i>=0;i--) {
                r = z*r + qr3[i];
                s = z*s + qs3[i];
            }
        } else if(x>2.5L) {
            r = qr4[12]; s = qs4[12];
            for(i=11;i>=0;i--) {
                r = z*r + qr4[i];
                s = z*s + qs4[i];
            }
        } else if(x> (1.0L/0.5625L)) {
```



```
            r = qr5[12]; s = qs5[12];
            for(i=11;i>=0;i--) {
                r = z*r + qr5[i];
                s = z*s + qs5[i];
            }
        } else {        /* assume x > 1.28 */
            r = qr6[12]; s = qs6[12];
            for(i=11;i>=0;i--) {
                r = z*r + qr6[i];
                s = z*s + qs6[i];
            }
        }
        return t*(r/s);
}
```

```
**********************************************************
    7031 Sun May  4 03:05:29 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"
#endif /* ! codereview */
```



```
#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 */
```

```
128                 b = powl(0.5L*x,(GENERIC) n);
129                 if (b!=zero) {
130                     for(a=one,i=1;i<=n;i++) a *= (GENERIC)i;
131                     b = b/a;
132                 }
133             } else {
134                 /* use backward recurrence */
135                 /*                         x      x^2      x^2
136                  *  J(n,x)/J(n-1,x) =  ----   ------   ------   .....
137                  *                         2n  - 2(n+1) - 2(n+2)
138                  *
139                  *                         1      1        1
140                  *  (for large x)   =  ----  ------   ------   .....
141                  *                         2n   2(n+1)   2(n+2)
142                  *                         -- - ------ - ------ -
143                  *                          x     x         x
144                  *
145                  * Let w = 2n/x and h=2/x, then the above quotient
146                  * is equal to the continued fraction:
147                  *                   1
148                  *      = -----------------------
149                  *                  1
150                  *         w - -----------------
151                  *                       1
152                  *              w+h - ---------
153                  *                    w+2h - ...
154                  *
155                  * To determine how many terms needed, let
156                  * Q(0) = w, Q(1) = w(w+h) - 1,
157                  * Q(k) = (w+k*h)*Q(k-1) - Q(k-2),
158                  * When Q(k) > 1e4      good for single
159                  * When Q(k) > 1e9      good for double
160                  * When Q(k) > 1e17     good for quaduple
161                  */
162             /* determin k */
163                 GENERIC t,v;
164                 double q0,q1,h,tmp; int k,m;
165                 w  = (n+n)/(double)x; h = 2.0/(double)x;
166                 q0 = w;  z = w+h; q1 = w*z - 1.0; k=1;
167                 while(q1<1.0e17) {
168                         k += 1; z += h;
169                         tmp = z*q1 - q0;
170                         q0 = q1;
171                         q1 = tmp;
172                 }
173                 m = n+n;
174                 for(t=zero, i = 2*(n+k); i>=m; i -= 2) t = one/(i/x-t);
175                 a = t;
176                 b = one;
177                 /* estimate log((2/x)^n*n!) = n*log(2/x)+n*ln(n)
178                    hence, if n*(log(2n/x)) > ...
179                     single 8.8722839355e+01
180                     double 7.09782712893383973096e+02
181                     long double 1.135652340629414394949193107797076500617 0e+04
182                     then recurrent value may overflow and the result is
183                     likely underflow to zero
184                  */
185                 tmp = n;
186                 v = two/x;
187                 tmp = tmp*logl(fabsl(v*tmp));
188                 if(tmp<1.135652340629414394949193107797076500 4L) {
189                     for(i=n-1;i>0;i--){
190                         temp = b;
191                         b = ((i+i)/x)*b - a;
192                         a = temp;
193                     }
```



```
194                 } else {
195                     for(i=n-1;i>0;i--){
196                         temp = b;
197                         b = ((i+i)/x)*b - a;
198                         a = temp;
199                         if(b>1e1000L) {
200                             a /= b;
201                             t /= b;
202                             b  = 1.0;
203                         }
204                     }
205                 }
206                 b = (t*j0l(x)/b);
207             }
208         }
209         if(sgn==1) return -b; else return b;
210 }

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

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

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

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

```
257                 a = y0l(x);
258                 b = y1l(x);
259                 /*
260                  * fix 1262058 and take care of non-default rounding
261                  */
262                 for (i = 1; i < n; i++) {
263                         temp = b;
264                         b *= (GENERIC) (i + i) / x;
265                         if (b <= -LDBL_MAX)
266                                 break;
267                         b -= a;
268                         a = temp;
269                 }
270         }
271         if(sign>0) return b; else return -b;
272 }
```

```
**********************************************************
    1619 Sun May  4 03:05:31 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  4 03:05:33 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_***

```
**********************************************************
    2927 Sun May  4 03:05:34 2014
new/usr/src/lib/libm/common/LD/sincosl.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 sincosl = __sincosl

/* INDENT OFF */
/* cosl(x)
 * Table look-up algorithm by K.C. Ng, November, 1989.
 *
 * kernel function:
 *      __k_sincosl     ... sin and cos function on [-pi/4,pi/4]
 *      __rem_pio2l     ... argument reduction routine
 *
 * Method.
 *      Let S and C denote the sin and cos respectively on [-PI/4, +PI/4].
 *      1. Assume the argument x is reduced to y1+y2 = x-k*pi/2 in
 *         [-pi/2 , +pi/2], and let n = k mod 4.
 *      2. Let S=S(y1+y2), C=C(y1+y2). Depending on n, we have
 *
 *          n        sin(x)      cos(x)        tan(x)
 *     ----------------------------------------------------------
 *          0          S           C             S/C
 *          1          C          -S            -C/S
 *          2         -S          -C             S/C
 *          3         -C           S            -C/S
 *     ----------------------------------------------------------
 *
 * Special cases:
 *      Let trig be any of sin, cos, or tan.
 *      trig(+-INF)  is NaN, with signals;
 *      trig(NaN)    is that NaN;
 *
 * Accuracy:
 *      computer TRIG(x) returns trig(x) nearly rounded.
 */
/* INDENT ON */
```



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

#include <sys/isa_defs.h>

void
sincosl(long double x, long double *s, long double *c) {
        long double y[2], z = 0.0L;
        int n, ix;
#if defined(__i386) || defined(__amd64)
#if defined(_LITTLE_ENDIAN)
        int *px = (int *) &x;
#endif

        /* trig(Inf or NaN) is NaN */
        if (!finitel(x)) {
                *s = *c = x - x;
                return;
        }

        /* High word of x. */
#if defined(__i386) || defined(__amd64)
        XTOI(px, ix);
#else
#if defined(_BIG_ENDIAN)
        ix = *(int *) &x;
#else
        XTOI(px, ix);
#endif

        /* |x| ~< pi/4 */
        ix &= 0x7fffffff;
        if (ix <= 0x3ffe9220)
                *s = __k_sincosl(x, z, c);

        /* argument reduction needed */
        else {
                n = __rem_pio2l(x, y);
                switch (n & 3) {
                case 0:
                        *s = __k_sincosl(y[0], y[1], c);
                        break;
                case 1:
                        *c = -__k_sincosl(y[0], y[1], s);
                        break;
                case 2:
                        *s = -__k_sincosl(y[0], y[1], c);
                        *c = -*c;
                        break;
                case 3:
                        *c = __k_sincosl(y[0], y[1], s);
                        *s = -*s;
                }
        }
}
```

_____*unchanged_portion_omitted_*

```
**********************************************************
    6047 Sun May  4 03:05:36 2014
new/usr/src/lib/libm/common/LD/sincospil.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 sincospil = __sincospil

/*
 * void sincospil(long double x, long double *s, long double *c)
 * *s = sinl(pi*x); *c = cosl(pi*x);
 *
 * Algorithm, 10/17/2002, K.C. Ng
 * ------------------------------
 * Let y = |4x|, z = floor(y), and n = (int)(z mod 8.0) (displayed in binary).
 *      1. If y==z, then x is a multiple of pi/4. Return the following values:
 *              ---------------------------------------------------
 *                n  x mod 2    sin(x*pi)    cos(x*pi)   tan(x*pi)
 *              ---------------------------------------------------
 *               000  0.00       +0 ___       +1 ___      +0
 *               001  0.25       +\/0.5       +\/0.5      +1
 *               010  0.50       +1 ___       +0 ___      +inf
 *               011  0.75       +\/0.5       -\/0.5      -1
 *               100  1.00       -0 ___       -1 ___      +0
 *               101  1.25       -\/0.5       -\/0.5      +1
 *               110  1.50       -1 ___       -0 ___      +inf
 *               111  1.75       -\/0.5       +\/0.5      -1
 *              ---------------------------------------------------
 *      2. Otherwise,
 *             ---------------------------------------------------
 *               n     t        sin(x*pi)    cos(x*pi)   tan(x*pi)
 *             ---------------------------------------------------
 *              000  (y-z)/4     sinpi(t)     cospi(t)    tanpi(t)
 *              001  (z+1-y)/4   cospi(t)     sinpi(t)    1/tanpi(t)
 *              010  (y-z)/4     cospi(t)    -sinpi(t)   -1/tanpi(t)
 *              011  (z+1-y)/4   sinpi(t)    -cospi(t)   -tanpi(t)
 *              100  (y-z)/4    -sinpi(t)    -cospi(t)    tanpi(t)
 *              101  (z+1-y)/4  -cospi(t)    -sinpi(t)    1/tanpi(t)
 *              110  (y-z)/4    -cospi(t)     sinpi(t)   -1/tanpi(t)
```



```
 *              111  (z+1-y)/4  -sinpi(t)     cospi(t)   -tanpi(t)
 *             ---------------------------------------------------
 *
 * NOTE. This program compute sinpi/cospi(t<0.25) by __k_sin/cos(pi*t, 0.0).
 * This will return a result with error slightly more than one ulp (but less
 * than 2 ulp). If one wants accurate result,  one may break up pi*t in
 * high (tpi_h) and low (tpi_l) parts and call __k_sin/cos(tip_h, tip_lo)
 * instead.
 */

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

#include <sys/isa_defs.h>

#define I(q, m) ((int *) &(q))[m]
#define U(q, m) ((unsigned *) &(q))[m]
#if defined(__i386) || defined(__amd64)
#if defined(_LITTLE_ENDIAN)
#define LDBL_MOST_SIGNIF_I(ld)  ((I(ld, 2) << 16) | (0xffff & (I(ld, 1) >> 15)))
#define LDBL_LEAST_SIGNIF_U(ld) U(ld, 0)
#define PREC    64
#define PRECM1  63
#define PRECM2  62
static const long double twoPRECM2 = 9.22337203685477580800000000000000e+18L;
#else
#define LDBL_MOST_SIGNIF_I(ld)  I(ld, 0)
#define LDBL_LEAST_SIGNIF_U(ld) U(ld, sizeof(long double) / sizeof(int) - 1)
#define PREC    113
#define PRECM1  112
#define PRECM2  111
static const long double twoPRECM2 = 5.19229685853482762853049632922009 6e+33L;
#endif

static const long double
zero    = 0.0L,
quater  = 0.25L,
one     = 1.0L,
pi      = 3.14159265358979323846264338327950288419 7e+0000L,
sqrth   = 0.70710678118654752440084436210484903928 4835937688474,
tiny    = 1.0e-100;

void
sincospil(long double x, long double *s, long double *c) {
        long double y, z, t;
        int hx, n, k;
        unsigned lx;

        hx = LDBL_MOST_SIGNIF_I(x);
        lx = LDBL_LEAST_SIGNIF_U(x);
        k = ((hx & 0x7fff0000) >> 16) - 0x3fff;
        if (k >= PRECM2) {              /* |x| >= 2**(Prec-2) */
                if (k >= 16384) {
                        *s = *c = x - x;
                }
                else {
                        if (k >= PREC) {
                                *s = zero;
                                *c = one;
                        }
                        else if (k == PRECM1) {
                                if ((lx & 1) == 0) {
                                        *s = zero;
                                        *c = one;
                                }
```

```
128                                 else {
129                                         *s = -zero;
130                                         *c = -one;
131                                 }
132                         }
133                         else {  /* k = Prec - 2 */
134                                 if ((lx & 1) == 0) {
135                                         *s = zero;
136                                         *c = one;
137                                 }
138                                 else {
139                                         *s = one;
140                                         *c = zero;
141                                 }
142                                 if ((lx & 2) != 0) {
143                                         *s = -*s;
144                                         *c = -*c;
145                                 }
146                         }
147                 }
148         }
149         else if (k < -2)        /* |x| < 0.25 */
150                 *s = __k_sincosl(pi * fabsl(x), zero, c);
151         else {
152                 /* y = |4x|, z = floor(y), and n = (int)(z mod 8.0) */
153                 y = 4.0L * fabsl(x);
154                 if (k < PRECM2) {
155                         z = y + twoPRECM2;
156                         n = LDBL_LEAST_SIGNIF_U(z) & 7; /* 3 LSb of z */
157                         t = z - twoPRECM2;
158                         k = 0;
159                         if (t == y)
160                                 k = 1;
161                         else if (t > y) {
162                                 n -= 1;
163                                 t = quater + (y - t) * quater;
164                         }
165                         else
166                                 t = (y - t) * quater;
167                 }
168                 else {  /* k = Prec-3 */
169                         n = LDBL_LEAST_SIGNIF_U(y) & 7; /* 3 LSb of z */
170                         k = 1;
171                 }
172                 if (k) {        /* x = N/4 */
173                         if((n & 1) != 0)
174                                 *s = *c = sqrth + tiny;
175                         else
176                                 if ((n & 2) == 0) {
177                                         *s = zero;
178                                         *c = one;
179                                 }
180                                 else {
181                                         *s = one;
182                                         *c = zero;
183                                 }
184                         if ((n & 4) != 0)
185                                 *s = -*s;
186                         if (((n + 1) & 4) != 0)
187                                 *c = -*c;
188                 }
189                 else {
190                         if ((n & 1) != 0)
191                                 t = quater - t;
192                         if (((n + (n & 1)) & 2) == 0)
193                                 *s = __k_sincosl(pi * t, zero, c);
```



```
194                         else
195                                 *c = __k_sincosl(pi * t, zero, s);
196                         if ((n & 4) != 0)
197                                 *s = -*s;
198                         if (((n + 2) & 4) != 0)
199                                 *c = -*c;
200                 }
201         }
202         if (hx < 0)
203                 *s = -*s;
204 }
```

unchanged_portion_omitted_

```
**********************************************************
    2239 Sun May  4 03:05:37 2014
new/usr/src/lib/libm/common/LD/sinhl.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 sinhl = __sinhl

#include "libm.h"
#include "longdouble.h"
#endif /* ! codereview */

/* SINH(X)
 * RETURN THE HYPERBOLIC SINE OF X
 *
 * Method :
 *      1. reduce x to non-negative by SINH(-x) = - SINH(x).
 *      2.
 *                                                 EXPM1(x) + EXPM1(x)/(EXPM1(x)+1)
 *          0 <= x <= lnovft     : SINH(x) := -------------------------------
 *                                                            2
 *
 *     lnovft <= x <  INF        : SINH(x) := EXP(x-MEP1*ln2)*2**ME
 *
 * here
 *      lnovft          logarithm of the overflow threshold
 *                      = MEP1*ln2 chopped to machine precision.
 *      ME              maximum exponent
 *      MEP1            maximum exponent plus 1
 *
 * Special cases:
 *      SINH(x) is x if x is +INF, -INF, or NaN.
 *      only SINH(0)=0 is exact for finite argument.
 *
 */

static const long double C[] = {
        0.5L,
```



```
        1.0L,
        1.13565234062941439487914 9e+04L,
        7.00444768624254908785898 5e-16L
};

#define half    C[0]
#define one     C[1]
#define lnovft  C[2]
#define lnovlo  C[3]

long double
sinhl(long double x)
{
        long double     r, t;

        if (!finitel(x))
                return (x + x); /* x is INF or NaN */
        r = fabsl(x);
        if (r < lnovft) {
                t = expm1l(r);
                r = copysignl((t + t / (one + t)) * half, x);
        } else {
                r = copysignl(expl((r - lnovft) - lnovlo), x);
                r = scalbnl(r, 16383);
        }
        return (r);
}
```

```
**********************************************************
    2866 Sun May  4 03:05:38 2014
new/usr/src/lib/libm/common/LD/sinl.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 sinl = __sinl

/* INDENT OFF */
/* sinl(x)
 * Table look-up algorithm by K.C. Ng, November, 1989.
 *
 * kernel function:
 *      __k_sinl               ... sin function on [-pi/4,pi/4]
 *      __k_cosl               ... cos function on [-pi/4,pi/4]
 *      __rem_pio2l     ... argument reduction routine
 *
 * Method.
 *      Let S and C denote the sin and cos respectively on [-PI/4, +PI/4].
 *      1. Assume the argument x is reduced to y1+y2 = x-k*pi/2 in
 *         [-pi/2 , +pi/2], and let n = k mod 4.
 *      2. Let S=S(y1+y2), C=C(y1+y2). Depending on n, we have
 *
 *          n        sin(x)      cos(x)        tan(x)
 *     ----------------------------------------------------------
 *          0          S           C             S/C
 *          1          C          -S            -C/S
 *          2         -S          -C             S/C
 *          3         -C           S            -C/S
 *     ----------------------------------------------------------
 *
 * Special cases:
 *      Let trig be any of sin, cos, or tan.
 *      trig(+-INF)  is NaN, with signals;
 *      trig(NaN)    is that NaN;
 *
 * Accuracy:
 *      computer TRIG(x) returns trig(x) nearly rounded.
 */
```



```
/* INDENT ON */

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

#include <sys/isa_defs.h>

long double
sinl(long double x) {
        long double y[2], z = 0.0L;
        int n, ix;
#if defined(__i386) || defined(__amd64)
#if defined(_LITTLE_ENDIAN)
        int *px = (int *) &x;
#endif

        /* sin(Inf or NaN) is NaN */
        if (!finitel(x))
                return x - x;

        /* High word of x. */
#if defined(__i386) || defined(__amd64)
        XTOI(px, ix);
#else
#if defined(_BIG_ENDIAN)
        ix = *(int *) &x;
#else
        XTOI(px, ix);
#endif
        /* |x| ~< pi/4 */
        ix &= 0x7fffffff;
        if (ix <= 0x3ffe9220)
        if (ix <= 0x3ffe9220) {
                return __k_sinl(x, z);
        }

        /* argument reduction needed */
        else {
                n = __rem_pio2l(x, y);
                switch (n & 3) {
                case 0:
                        return __k_sinl(y[0], y[1]);
                case 1:
                        return __k_cosl(y[0], y[1]);
                case 2:
                        return -__k_sinl(y[0], y[1]);
                case 3:
                        return -__k_cosl(y[0], y[1]);
                /* NOTREACHED */
                }
        }
        return 0.0L;
}
```

_____unchanged_portion_omitted_

```
**********************************************************
    5594 Sun May  4 03:05:39 2014
new/usr/src/lib/libm/common/LD/sinpil.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 sinpil = __sinpil

/* long double sinpil(long double x),
 * return long double precision sinl(pi*x).
 *
 * Algorithm, 10/17/2002, K.C. Ng
 * ------------------------------
 * Let y = |4x|, z = floor(y), and n = (int)(z mod 8.0) (displayed in binary).
 *      1. If y==z, then x is a multiple of pi/4. Return the following values:
 *              ---------------------------------------------------
 *                n  x mod 2    sin(x*pi)    cos(x*pi)   tan(x*pi)
 *              ---------------------------------------------------
 *               000  0.00       +0 ___       +1 ___      +0
 *               001  0.25       +\/0.5       +\/0.5      +1
 *               010  0.50       +1 ___       +0 ___      +inf
 *               011  0.75       +\/0.5       -\/0.5      -1
 *               100  1.00       -0 ___       -1 ___      +0
 *               101  1.25       -\/0.5       -\/0.5      +1
 *               110  1.50       -1 ___       -0 ___      +inf
 *               111  1.75       -\/0.5       +\/0.5      -1
 *              ---------------------------------------------------
 *      2. Otherwise,
 *              ---------------------------------------------------
 *                n     t        sin(x*pi)    cos(x*pi)   tan(x*pi)
 *              ---------------------------------------------------
 *               000  (y-z)/4     sinpi(t)     cospi(t)    tanpi(t)
 *               001  (z+1-y)/4   cospi(t)     sinpi(t)    1/tanpi(t)
 *               010  (y-z)/4     cospi(t)    -sinpi(t)   -1/tanpi(t)
 *               011  (z+1-y)/4   sinpi(t)    -cospi(t)   -tanpi(t)
 *               100  (y-z)/4    -sinpi(t)    -cospi(t)    tanpi(t)
 *               101  (z+1-y)/4  -cospi(t)    -sinpi(t)    1/tanpi(t)
 *               110  (y-z)/4    -cospi(t)     sinpi(t)   -1/tanpi(t)
 *               111  (z+1-y)/4  -sinpi(t)     cospi(t)   -tanpi(t)
```



```
 *              ---------------------------------------------------
 *
 * NOTE. This program compute sinpi/cospi(t<0.25) by __k_sin/cos(pi*t, 0.0).
 * This will return a result with error slightly more than one ulp (but less
 * than 2 ulp). If one wants accurate result,  one may break up pi*t in
 * high (tpi_h) and low (tpi_l) parts and call __k_sin/cos(tip_h, tip_lo)
 * instead.
 */

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

#include <sys/isa_defs.h>

#define I(q, m) ((int *) &(q))[m]
#define U(q, m) ((unsigned *) &(q))[m]
#if defined(__i386) || defined(__amd64)
#if defined(_LITTLE_ENDIAN)
#define LDBL_MOST_SIGNIF_I(ld)  ((I(ld, 2) << 16) | (0xffff & (I(ld, 1) >> 15)))
#define LDBL_LEAST_SIGNIF_U(ld) U(ld, 0)
#define PREC    64
#define PRECM1  63
#define PRECM2  62
static const long double twoPRECM2 = 9.22337203685477580800000000000000000e+18L;
#else
#define LDBL_MOST_SIGNIF_I(ld)  I(ld, 0)
#define LDBL_LEAST_SIGNIF_U(ld) U(ld, sizeof(long double) / sizeof(int) - 1)
#define PREC    113
#define PRECM1  112
#define PRECM2  111
static const long double twoPRECM2 = 5.19229685853482762853049632922009 6e+33L;
#endif

static const long double
zero    = 0.0L,
quater  = 0.25L,
one     = 1.0L,
pi      = 3.14159265358979323846264338327950288419 7e+0000L,
sqrth   = 0.70710678118654752440084436210484903928 4835937688474,
tiny    = 1.0e-100;

long double
sinpil(long double x) {
        long double y, z, t;
        int hx, n, k;
        unsigned lx;

        hx = LDBL_MOST_SIGNIF_I(x);
        lx = LDBL_LEAST_SIGNIF_U(x);
        k = ((hx & 0x7fff0000) >> 16) - 0x3fff;
        if (k >= PRECM2) {              /* |x| >= 2**(Prec-2) */
                if (k >= 16384)
                        y = x - x;
                else {
                        if (k >= PREC)
                                y = zero;
                        else if (k == PRECM1)
                                y = (lx & 1) == 0 ? zero: -zero;
                        else {  /* k = Prec - 2 */
                                y = (lx & 1) == 0 ? zero : one;
                                if ((lx & 2) != 0)
                                        y = -y;
                        }
                }
        }
```

```
128         else if (k < -2)        /* |x| < 0.25 */
129                 y = __k_sinl(pi * fabsl(x), zero);
130         else {
131                 /* y = |4x|, z = floor(y), and n = (int)(z mod 8.0) */
132                 y = 4.0L * fabsl(x);
133                 if (k < PRECM2) {
134                         z = y + twoPRECM2;
135                         n = LDBL_LEAST_SIGNIF_U(z) & 7; /* 3 LSb of z */
136                         t = z - twoPRECM2;
137                         k = 0;
138                         if (t == y)
139                                 k = 1;
140                         else if (t > y) {
141                                 n -= 1;
142                                 t = quater + (y - t) * quater;
143                         }
144                         else
145                                 t = (y - t) * quater;
146                 }
147                 else {  /* k = Prec-3 */
148                         n = LDBL_LEAST_SIGNIF_U(y) & 7; /* 3 LSb of z */
149                         k = 1;
150                 }
151                 if (k) {        /* x = N/4 */
152                         if((n & 1) != 0)
153                                 y = sqrth + tiny;
154                         else
155                                 y = (n & 2) == 0 ? zero : one;
156                         if ((n & 4) != 0)
157                                 y = -y;
158                 }
159                 else {
160                         if ((n & 1) != 0)
161                                 t = quater - t;
162                         if (((n + (n & 1)) & 2) == 0)
163                                 y = __k_sinl(pi * t, zero);
164                         else
165                                 y = __k_cosl(pi * t, zero);
166                         if ((n & 4) != 0)
167                                 y = -y;
168                 }
169         }
170         return hx >= 0 ? y : -y;
171 }
```

*_____unchanged_portion_omitted_*

****
2608 Sun May 4 03:05:41 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"
#endif /* ! codereview */

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_***

```
**********************************************************
    2640 Sun May  4 03:05:44 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.
 */

#pragma weak tanl = __tanl

/* INDENT OFF */
/* cosl(x)
 * Table look-up algorithm by K.C. Ng, November, 1989.
 *
 * kernel function:
 *      __k_tanl        ... tangent function on [-pi/4,pi/4]
 *      __rem_pio2l     ... argument reduction routine
 *
 * Method.
 *      Let S and C denote the sin and cos respectively on [-PI/4, +PI/4].
 *      1. Assume the argument x is reduced to y1+y2 = x-k*pi/2 in
 *         [-pi/2 , +pi/2], and let n = k mod 4.
 *      2. Let S=S(y1+y2), C=C(y1+y2). Depending on n, we have
 *
 *          n        sin(x)      cos(x)        tan(x)
 *     ----------------------------------------------------------
 *          0          S           C             S/C
 *          1          C          -S            -C/S
 *          2         -S          -C             S/C
 *          3         -C           S            -C/S
 *     ----------------------------------------------------------
 *
 * Special cases:
 *      Let trig be any of sin, cos, or tan.
 *      trig(+-INF)  is NaN, with signals;
 *      trig(NaN)    is that NaN;
 *
 * Accuracy:
 *      computer TRIG(x) returns trig(x) nearly rounded.
 */
/* INDENT ON */
```



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

#include <sys/isa_defs.h>

long double
tanl(long double x) {
        long double y[2], z = 0.0L;
        int n, ix;
#if defined(__i386) || defined(__amd64)
#if defined(_LITTLE_ENDIAN)
        int *px = (int *) &x;
#endif

        /* trig(Inf or NaN) is NaN */
        if (!finitel(x))
                return x - x;

        /* High word of x. */
#if defined(__i386) || defined(__amd64)
        XTOI(px, ix);
#else
#if defined(_BIG_ENDIAN)
        ix = *(int *) &x;
#else
        XTOI(px, ix);
#endif

        /* |x| ~< pi/4 */
        ix &= 0x7fffffff;
        if (ix <= 0x3ffe9220)
                return __k_tanl(x, z, 0);

        /* argument reduction needed */
        else {
                n = __rem_pio2l(x, y);
                return __k_tanl(y[0], y[1], n & 1);
        }
}
```

*_____unchanged_portion_omitted_*

```
**********************************************************
    5459 Sun May  4 03:05:46 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  4 03:05:47 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  4 03:05:49 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  4 03:05:50 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  4 03:05:52 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  4 03:05:53 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  4 03:05:55 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  4 03:05:57 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_
```

```
**********************************************************
    1231 Sun May  4 03:05:59 2014
new/usr/src/lib/libm/common/R/exp10f.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 exp10f = __exp10f

#include "libm.h"

extern double exp10(double);
double exp10(double);

float
exp10f(float x) {
#if defined(FPADD_TRAPS_INCOMPLETE_ON_NAN)
        if (isnanf(x))
                return (x * x);
        else
#endif
        return ((float) exp10((double) x));
}
______unchanged_portion_omitted_
```

```
**********************************************************
    5085 Sun May  4 03:06:01 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_
```

****
1385 Sun May 4 03:06:03 2014
new/usr/src/lib/libm/common/R/sincospif.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 sincospif = __sincospif

#include "libm.h"

extern void sincospi(double, double *, double *);
void sincospi(double x, double *s, double *c);

void
sincospif(float x, float *s, float *c) {
        double ds, dc;

#if defined(FPADD_TRAPS_INCOMPLETE_ON_NAN)
        if (isnanf(x))
                *s = *c = x * x;
        else {
#endif
        sincospi((double) x, &ds, &dc);
        *s = (float) ds;
        *c = (float) dc;
#if defined(FPADD_TRAPS_INCOMPLETE_ON_NAN)
        }
#endif
}
```

_____unchanged_portion_omitted_

```
**********************************************************
    3911 Sun May  4 03:06:04 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  4 03:06:06 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_
```

```
**********************************************************
    9501 Sun May  4 03:06:08 2014
new/usr/src/lib/libm/common/complex/cpow.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 cpow = __cpow

/* INDENT OFF */
/*
 * dcomplex cpow(dcomplex z);
 *
 * z**w analytically equivalent to
 *
 * cpow(z,w) = cexp(w clog(z))
 *
 * Let z = x+iy, w = u+iv.
 * Since
 *                                  _________
 *                                 / 2    2                -1   y
 *     log(x+iy) = log(\/ x  + y    ) + i tan   (---)
 *                                                          x
 *
 *                    1       2    2          -1   y
 *                = --- log(x  + y ) + i tan   (---)
 *                    2                       x
 *                     u       2    2          -1  y
 * (u+iv)* log(x+iy) =  --- log(x  + y ) - v tan  (---)  +          (1)
 *                      2                        x
 *
 *                           v       2    2          -1  y
 *                      i * [ --- log(x  + y ) + u tan  (---) ]      (2)
 *                            2                        x
 *
 *                   = r + i q
 *
 * Therefore,
 *      w     r+iq    r
 *     z  =  e     = e  (cos(q)+i*sin(q))
```



```
 *                                          _______
 *                                         / 2   2
 *       r                               \/ x + y     -v*atan2(y,x)
 * Here e  can be expressed as:  u          * e
 *
 * Special cases (in the order of appearance):
 *      1.  (anything) ** 0  is 1
 *      2.  (anything) ** 1  is itself
 *      3.  When v = 0, y = 0:
 *            If x is finite and negative, and u is finite, then
 *               x ** u = exp(u*pi i) * pow(|x|, u);
 *            otherwise,
 *               x ** u = pow(x, u);
 *      4.  When v = 0, x = 0 or |x| = |y| or x is inf or y is inf:
 *               (x + y i) ** u = r * exp(q i)
 *           where
 *               r = hypot(x,y) ** u
 *               q = u * atan2pi(y, x)
 *
 *      5.  otherwise, z**w is NAN if any x, y, u, v is a Nan or inf
 *
 *      Note: many results of special cases are obtained in terms of
 *      polar coordinate. In the conversion from polar to rectangle:
 *                   r exp(q i) = r * cos(q) + r * sin(q) i,
 *      we regard r * 0 is 0 except when r is a NaN.
 */
/* INDENT ON */

#include "libm.h"       /* atan2/exp/fabs/hypot/log/pow/scalbn */
                        /* atan2pi/exp2/sincos/sincospi/__k_clog_r/__k_atan2 */
#include "complex_wrapper.h"

extern void sincospi(double, double *, double *);

static const double
        huge = 1e300,
        tiny = 1e-300,
        invln2 = 1.44269504088896338700e+00,
        ln2hi = 6.93147180369123816490e-01,   /* 0x3fe62e42, 0xfee00000 */
        ln2lo = 1.90821492927058770002e-10,   /* 0x3dea39ef, 0x35793c76 */
        one = 1.0,
        zero = 0.0;

static const int hiinf = 0x7ff00000;
extern double atan2pi(double, double);
double atan2pi(double, double);

/*
 * Assuming |t[0]| > |t[1]| and |t[2]| > |t[3]|, sum4fp subroutine
 * compute t[0] + t[1] + t[2] + t[3] into two double fp numbers.
 */
static double
sum4fp(double ta[], double *w) {
        double t1, t2, t3, t4, w1, w2, t;
        t1 = ta[0]; t2 = ta[1]; t3 = ta[2]; t4 = ta[3];
        /*
         * Rearrange ti so that |t1| >= |t2| >= |t3| >= |t4|
         */
        if (fabs(t4) > fabs(t1)) {
                t = t1; t1 = t3; t3 = t;
                t = t2; t2 = t4; t4 = t;
        } else if (fabs(t3) > fabs(t1)) {
                t = t1; t1 = t3;
                if (fabs(t4) > fabs(t2)) {
                        t3 = t4; t4 = t2; t2 = t;
                } else {
```

```
128                         t3 = t2; t2 = t;
129                 }
130         } else if (fabs(t3) > fabs(t2)) {
131                 t = t2; t2 = t3;
132                 if (fabs(t4) > fabs(t2)) {
133                         t3 = t4; t4 = t;
134                 } else
135                         t3 = t;
136         }
137         /* summing r = t1 + t2 + t3 + t4 to w1 + w2 */
138         w1 = t3 + t4;
139         w2 = t4 - (w1 - t3);
140         t  = t2 + w1;
141         w2 += w1 - (t - t2);
142         w1 = t + w2;
143         w2 += t - w1;
144         t  = t1 + w1;
145         w2 += w1 - (t - t1);
146         w1 = t + w2;
147         *w = w2 - (w1 - t);
148         return (w1);
149 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
    5524 Sun May  4 03:06:09 2014
new/usr/src/lib/libm/common/complex/k_cexp.c
**********************************************************
______unchanged_portion_omitted_
```

```
 110 invln2 = 1.44269504088896338700e+00,    /* 0x3ff71547, 0x652b82fe */
 111 P1 = 1.66666666666660190370e-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 4 03:06:11 2014
new/usr/src/lib/libm/common/complex/k_clog_rl.c
****
_____unchanged_portion_omitted_

```
 408 long double
 409 __k_clog_rl(long double x, long double y, long double *er)
 410 {
 411         long double t1, t2, t3, t4, tk, z, wh, w, zh, zk;
 412         int n, k, ix, iy, iz, nx, ny, nz, i;
 413         double dk;

 415 #if !defined(__x86)
 416         int j;
 417         unsigned lx, ly;
 418 #endif

 420         ix = HI_XWORD(x) & ~0x80000000;
 421         iy = HI_XWORD(y) & ~0x80000000;
 422         y = fabsl(y); x = fabsl(x);
 423         if (ix < iy || (ix < 0x7fff0000 && ix == iy && x < y)) {
 424                 /* force x >= y */
 425                 tk = x;  x = y;   y = tk;
 426                 n = ix, ix = iy; iy = n;
 427         }
 428         *er = zero;
 429         nx = ix >> 16; ny = iy >> 16;
 430         if (nx >= 0x7fff) {     /* x or y is Inf or NaN */
 431                 if (isinfl(x))
 432                         return (x);
 433                 else if (isinfl(y))
 434                         return (y);
 435                 else
 436                         return (x+y);
 437         }
 438 /*
 439  * for tiny y:(double y < 2^-35, extended y < 2^-46, quad y < 2^-70)
 440  *
 441  *      log(sqrt(1 + y**2)) =  y**2 / 2 - y**4 / 8 + ...  =  y**2 / 2
 442  */
 443 #if defined(__x86)
 444         if (x == 1.0L && ny < (0x3fff - 46)) {
 445 #else
 446         if (x == 1.0L && ny < (0x3fff - 70)) {
 447 #endif

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

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

 457                         wh = (long double) dk;
 458                         *er = half * ((y - wh) * (y + wh) - (t2 - wh * wh));
 459                 }
 460                 return (half * t2);
 461         }
 462 /*
 463  * x or y is subnormal or zero
 464  */
 465         if (nx == 0) {
 466                 if (x == 0.0L)
 467                         return (-1.0L / x);
```



```
 468                 else {
 469                         x *= two240;
 470                         y *= two240;
 471                         ix = HI_XWORD(x);
 472                         iy = HI_XWORD(y);
 473                         nx = (ix >> 16) - 240;
 474                         ny = (iy >> 16) - 240;
 475                         /* guard subnormal flush to 0 */
 476                         if (x == 0.0L)
 477                                 return (-1.0L / x);
 478                 }
 479         } else if (ny == 0) {   /* y subnormal, scale it */
 480                 y *= two240;
 481                 iy = HI_XWORD(y);
 482                 ny = (iy >> 16) - 240;
 483         }
 484         n = nx - ny;
 485 /*
 486  * When y is zero or when x >> y, i.e.,  n > 62, 78, 122 for DBLE,
 487  * EXTENDED, QUAD respectively,
 488  * log(x) = log(sqrt(x * x + y * y)) to 27 extra bits.
 489  */

 491 #if defined(__x86)
 492         if (n > 78 || y == 0.0L) {
 493 #else
 494         if (n > 122 || y == 0.0L) {
 495 #endif

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

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

 507                 zh = (long double)dk;
 508                 k = i & 0x7f;   /* index of zk */
 509                 n = nx - 0x3fff;
 510                 *er = z - zh;
 511                 if (i == 0x80) {        /* if zk = 2.0, adjust scaling */
 512                         n += 1;
 513                         zh *= 0.5L;  *er *= 0.5L;
 514                 }
 515                 w = k_log_NKzl(n, k, zh, er);
 516         } else {
 517 /*
 518  * compute z = x*x + y*y
 519  */
 520                 XFSCALE(x, (0x3fff - (ix >> 16)));
 521                 XFSCALE(y, (0x3fff - n - (iy >> 16)));
 520                 XFSCALE(x, 0x3fff - (ix >> 16));
 521                 XFSCALE(y, 0x3fff - n - (iy >> 16));
 522                 ix = (ix & 0xffff) | 0x3fff0000;
 523                 iy = (iy & 0xffff) | (0x3fff0000 - (n << 16));
 524                 nx -= 0x3fff;
 525                 t1 = x * x; t2 = y * y;
 526                 wh = x;

 528 /* split x into correctly rounded half */
 529 #if defined(__x86)
 530                 ((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 4 03:06:12 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
 381 #define test_sse_hw     &_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  4 03:06:14 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  4 03:06:16 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 4 03:06:18 2014
new/usr/src/lib/libm/common/m9x/__fex_sse.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 "fenv_synonyms.h"
#include <ucontext.h>
#include <fenv.h>
#if defined(__SUNPRO_C)
#include <sunmath.h>
#else
#include <sys/ieeefp.h>
#endif
#include "fex_handler.h"
#include "fenv_inlines.h"

#if !defined(REG_PC)
#define REG_PC  EIP
#endif

#if !defined(REG_PS)
#define REG_PS  EFL
#endif

#ifdef __amd64
#define regno(X)        ((X < 4)? REG_RAX - X : \
                        ((X > 4)? REG_RAX + 1 - X : REG_RSP))
#else
#define regno(X)        (EAX - X)
#endif

/*
 * Support for SSE instructions
 */

/*
 * Decode an SSE instruction.  Fill in *inst and return the length of the
 * instruction in bytes.  Return 0 if the instruction is not recognized.
```



```
 */
int
__fex_parse_sse(ucontext_t *uap, sseinst_t *inst)
{
        unsigned char   *ip;
        char            *addr;
        int             i, dbl, simd, rex, modrm, sib, r;

        i = 0;
        ip = (unsigned char *)uap->uc_mcontext.gregs[REG_PC];

        /* look for pseudo-prefixes */
        dbl = 0;
        simd = SIMD;
        if (ip[i] == 0xF3) {
                simd = 0;
                i++;
        } else if (ip[i] == 0x66) {
                dbl = DOUBLE;
                i++;
        } else if (ip[i] == 0xF2) {
                dbl = DOUBLE;
                simd = 0;
                i++;
        }

        /* look for AMD64 REX prefix */
        rex = 0;
        if (ip[i] >= 0x40 && ip[i] <= 0x4F) {
                rex = ip[i];
                i++;
        }

        /* parse opcode */
        if (ip[i++] != 0x0F)
                return 0;
        switch (ip[i++]) {
        case 0x2A:
                inst->op = (int)cvtsi2ss + simd + dbl;
                if (!simd)
                        inst->op = (int)inst->op + (rex & 8);
                break;

        case 0x2C:
                inst->op = (int)cvttss2si + simd + dbl;
                if (!simd)
                        inst->op = (int)inst->op + (rex & 8);
                break;

        case 0x2D:
                inst->op = (int)cvtss2si + simd + dbl;
                if (!simd)
                        inst->op = (int)inst->op + (rex & 8);
                break;

        case 0x2E:
                /* oddball: scalar instruction in a SIMD opcode group */
                if (!simd)
                        return 0;
                inst->op = (int)ucomiss + dbl;
                break;

        case 0x2F:
                /* oddball: scalar instruction in a SIMD opcode group */
                if (!simd)
                        return 0;
```

```
129                 inst->op = (int)comiss + dbl;
130                 break;
131
132         case 0x51:
133                 inst->op = (int)sqrtss + simd + dbl;
134                 break;
135
136         case 0x58:
137                 inst->op = (int)addss + simd + dbl;
138                 break;
139
140         case 0x59:
141                 inst->op = (int)mulss + simd + dbl;
142                 break;
143
144         case 0x5A:
145                 inst->op = (int)cvtss2sd + simd + dbl;
146                 break;
147
148         case 0x5B:
149                 if (dbl) {
150                         if (simd)
151                                 inst->op = cvtps2dq;
152                         else
153                                 return 0;
154                 } else {
155                         inst->op = (simd)? cvtdq2ps : cvttps2dq;
156                 }
157                 break;
158
159         case 0x5C:
160                 inst->op = (int)subss + simd + dbl;
161                 break;
162
163         case 0x5D:
164                 inst->op = (int)minss + simd + dbl;
165                 break;
166
167         case 0x5E:
168                 inst->op = (int)divss + simd + dbl;
169                 break;
170
171         case 0x5F:
172                 inst->op = (int)maxss + simd + dbl;
173                 break;
174
175         case 0xC2:
176                 inst->op = (int)cmpss + simd + dbl;
177                 break;
178
179         case 0xE6:
180                 if (simd) {
181                         if (dbl)
182                                 inst->op = cvttpd2dq;
183                         else
184                                 return 0;
185                 } else {
186                         inst->op = (dbl)? cvtpd2dq : cvtdq2pd;
187                 }
188                 break;
189
190         default:
191                 return 0;
192         }
193
194         /* locate operands */
```



```
195         modrm = ip[i++];
196
197         if (inst->op == cvtss2si || inst->op == cvttss2si ||
198             inst->op == cvtsd2si || inst->op == cvttsd2si ||
199             inst->op == cvtss2siq || inst->op == cvttss2siq ||
200             inst->op == cvtsd2siq || inst->op == cvttsd2siq) {
201                 /* op1 is a gp register */
202                 r = ((rex & 4) << 1) | ((modrm >> 3) & 7);
203                 inst->op1 = (sseoperand_t *)&uap->uc_mcontext.gregs[regno(r)];
204         } else if (inst->op == cvtps2pi || inst->op == cvttps2pi ||
205             inst->op == cvtpd2pi || inst->op == cvttpd2pi) {
206                 /* op1 is a mmx register */
207 #ifdef __amd64
208                 inst->op1 = (sseoperand_t *)&uap->uc_mcontext.fpregs.fp_reg_set.
209                     fpchip_state.st[(modrm >> 3) & 7];
210 #else
211                 inst->op1 = (sseoperand_t *)(10 * ((modrm >> 3) & 7) +
212                     (char *)&uap->uc_mcontext.fpregs.fp_reg_set.
213                     fpchip_state.state[7]);
214 #endif
215         } else {
216                 /* op1 is a xmm register */
217                 r = ((rex & 4) << 1) | ((modrm >> 3) & 7);
218                 inst->op1 = (sseoperand_t *)&uap->uc_mcontext.fpregs.
219                     fp_reg_set.fpchip_state.xmm[r];
220         }
221
222         if ((modrm >> 6) == 3) {
223                 if (inst->op == cvtsi2ss || inst->op == cvtsi2sd ||
224                     inst->op == cvtsi2ssq || inst->op == cvtsi2sdq) {
225                         /* op2 is a gp register */
226                         r = ((rex & 1) << 3) | (modrm & 7);
227                         inst->op2 = (sseoperand_t *)&uap->uc_mcontext.
228                             gregs[regno(r)];
229                 } else if (inst->op == cvtpi2ps || inst->op == cvtpi2pd) {
230                         /* op2 is a mmx register */
231 #ifdef __amd64
232                         inst->op2 = (sseoperand_t *)&uap->uc_mcontext.fpregs.
233                             fp_reg_set.fpchip_state.st[modrm & 7];
234 #else
235                         inst->op2 = (sseoperand_t *)(10 * (modrm & 7) +
236                             (char *)&uap->uc_mcontext.fpregs.fp_reg_set.
237                             fpchip_state.state[7]);
238 #endif
239                 } else {
240                         /* op2 is a xmm register */
241                         r = ((rex & 1) << 3) | (modrm & 7);
242                         inst->op2 = (sseoperand_t *)&uap->uc_mcontext.fpregs.
243                             fp_reg_set.fpchip_state.xmm[r];
244                 }
245         } else if ((modrm & 0xc7) == 0x05) {
246 #ifdef __amd64
246 #if defined(__amd64)
247                 /* address of next instruction + offset */
248                 r = i + 4;
249                 if (inst->op == cmpss || inst->op == cmpps ||
250                     inst->op == cmpsd || inst->op == cmppd)
251                         r++;
252                 inst->op2 = (sseoperand_t *)(ip + r + *(int *)(ip + i));
253 #else
254                 /* absolute address */
255                 inst->op2 = (sseoperand_t *)(*(int *)(ip + i));
256 #endif
257                 i += 4;
258         } else {
259                 /* complex address */
```

```
260                 if ((modrm & 7) == 4) {
261                         /* parse sib byte */
262                         sib = ip[i++];
263                         if ((sib & 7) == 5 && (modrm >> 6) == 0) {
264                                 /* start with absolute address */
265                                 addr = (char *)(uintptr_t)(*(int *)(ip + i));
265                                 addr = (char *)(uintptr_t)(ip + i);
266                                 i += 4;
267                         } else {
268                                 /* start with base */
269                                 r = ((rex & 1) << 3) | (sib & 7);
270                                 addr = (char *)uap->uc_mcontext.gregs[regno(r)];
271                         }
272                         r = ((rex & 2) << 2) | ((sib >> 3) & 7);
273                         if (r != 4) {
274                                 /* add scaled index */
275                                 addr += uap->uc_mcontext.gregs[regno(r)]
276                                     << (sib >> 6);
277                         }
278                 } else {
279                         r = ((rex & 1) << 3) | (modrm & 7);
280                         addr = (char *)uap->uc_mcontext.gregs[regno(r)];
281                 }

283                 /* add displacement, if any */
284                 if ((modrm >> 6) == 1) {
285                         addr += (char)ip[i++];
286                 } else if ((modrm >> 6) == 2) {
287                         addr += *(int *)(ip + i);
288                         i += 4;
289                 }
290                 inst->op2 = (sseoperand_t *)addr;
291         }

293         if (inst->op == cmpss || inst->op == cmpps || inst->op == cmpsd ||
294             inst->op == cmppd) {
295                 /* get the immediate operand */
296                 inst->imm = ip[i++];
297         }

299         return i;
300 }
```

*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;

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;

745             case sqrtss:
746                     info->op = fex_sqrt;
747                     sse_sqrtss(&info->op1.val.f, &info->res.val.f);
748                     break;

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;
```



```
756             case cvtsi2ss:
757                     info->op = fex_cnvt;
758                     sse_cvtsi2ss(&info->op1.val.i, &info->res.val.f);
759                     break;

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;

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;

773 #ifdef __amd64
774             case cvtsi2ssq:
775                     info->op = fex_cnvt;
776                     sse_cvtsi2ssq(&info->op1.val.l, &info->res.val.f);
777                     break;

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;

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

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;

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);

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 }

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;

848         e[0] = e[1] = e[2] = e[3] = -1;

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++) {
992                         dummy.op1 = (sseoperand_t *)&inst->op1->d[i];
993                         dummy.op2 = (sseoperand_t *)&inst->op2->d[i];
994                         e[i] = __fex_get_sse_op(uap, &dummy, &info[i]);
995                 }
996                 break;

998         case maxpd:
999                 dummy.op = maxsd;
1000                for (i = 0; i < 2; i++) {
1001                        dummy.op1 = (sseoperand_t *)&inst->op1->d[i];
1002                        dummy.op2 = (sseoperand_t *)&inst->op2->d[i];
1003                        e[i] = __fex_get_sse_op(uap, &dummy, &info[i]);
1004                }
1005                break;

1007        case addpd:
1008                dummy.op = addsd;
1009                for (i = 0; i < 2; i++) {
1010                        dummy.op1 = (sseoperand_t *)&inst->op1->d[i];
1011                        dummy.op2 = (sseoperand_t *)&inst->op2->d[i];
1012                        e[i] = __fex_get_sse_op(uap, &dummy, &info[i]);
1013                }
1014                break;

1016        case subpd:
1017                dummy.op = subsd;
1018                for (i = 0; i < 2; i++) {
1019                        dummy.op1 = (sseoperand_t *)&inst->op1->d[i];
```



```
1020                        dummy.op2 = (sseoperand_t *)&inst->op2->d[i];
1021                        e[i] = __fex_get_sse_op(uap, &dummy, &info[i]);
1022                }
1023                break;

1025        case mulpd:
1026                dummy.op = mulsd;
1027                for (i = 0; i < 2; i++) {
1028                        dummy.op1 = (sseoperand_t *)&inst->op1->d[i];
1029                        dummy.op2 = (sseoperand_t *)&inst->op2->d[i];
1030                        e[i] = __fex_get_sse_op(uap, &dummy, &info[i]);
1031                }
1032                break;

1034        case divpd:
1035                dummy.op = divsd;
1036                for (i = 0; i < 2; i++) {
1037                        dummy.op1 = (sseoperand_t *)&inst->op1->d[i];
1038                        dummy.op2 = (sseoperand_t *)&inst->op2->d[i];
1039                        e[i] = __fex_get_sse_op(uap, &dummy, &info[i]);
1040                }
1041                break;

1043        case sqrtpd:
1044                dummy.op = sqrtsd;
1045                for (i = 0; i < 2; i++) {
1046                        dummy.op1 = (sseoperand_t *)&inst->op1->d[i];
1047                        dummy.op2 = (sseoperand_t *)&inst->op2->d[i];
1048                        e[i] = __fex_get_sse_op(uap, &dummy, &info[i]);
1049                }
1050                break;

1052        case cvtpi2pd:
1053        case cvtdq2pd:
1054                dummy.op = cvtsi2sd;
1055                for (i = 0; i < 2; i++) {
1056                        dummy.op1 = (sseoperand_t *)&inst->op1->d[i];
1057                        dummy.op2 = (sseoperand_t *)&inst->op2->i[i];
1058                        e[i] = __fex_get_sse_op(uap, &dummy, &info[i]);
1059                }
1060                break;

1062        case cvttpd2pi:
1063        case cvttpd2dq:
1064                dummy.op = cvttsd2si;
1065                for (i = 0; i < 2; i++) {
1066                        dummy.op1 = (sseoperand_t *)&inst->op1->i[i];
1067                        dummy.op2 = (sseoperand_t *)&inst->op2->d[i];
1068                        e[i] = __fex_get_sse_op(uap, &dummy, &info[i]);
1069                }
1070                break;

1072        case cvtpd2pi:
1073        case cvtpd2dq:
1074                dummy.op = cvtsd2si;
1075                for (i = 0; i < 2; i++) {
1076                        dummy.op1 = (sseoperand_t *)&inst->op1->i[i];
1077                        dummy.op2 = (sseoperand_t *)&inst->op2->d[i];
1078                        e[i] = __fex_get_sse_op(uap, &dummy, &info[i]);
1079                }
1080                break;

1082        case cvtps2pd:
1083                dummy.op = cvtss2sd;
1084                for (i = 0; i < 2; i++) {
1085                        dummy.op1 = (sseoperand_t *)&inst->op1->d[i];
```

```
1086                        dummy.op2 = (sseoperand_t *)&inst->op2->f[i];
1087                        e[i] = __fex_get_sse_op(uap, &dummy, &info[i]);
1088                }
1089                break;

1091        case cvtpd2ps:
1092                dummy.op = cvtsd2ss;
1093                for (i = 0; i < 2; i++) {
1094                        dummy.op1 = (sseoperand_t *)&inst->op1->f[i];
1095                        dummy.op2 = (sseoperand_t *)&inst->op2->d[i];
1096                        e[i] = __fex_get_sse_op(uap, &dummy, &info[i]);
1097                }
1098        default:
1099                break;
1100 #endif /* ! codereview */
1101        }
1102 }

1104 /*
1105  * Store the result value from *info in the destination of the scalar
1106  * SSE instruction specified by *inst.  If no result is given but the
1107  * exception is underflow or overflow, supply the default trapped result.
1108  *
1109  * This routine does not work if the instruction specified by *inst
1110  * is not a scalar instruction.
1111  */
1112 void
1113 __fex_st_sse_result(ucontext_t *uap, sseinst_t *inst, enum fex_exception e,
1114     fex_info_t *info)
1115 {
1116        int             i = 0;
1117        long long       l = 0L;;
1118        float           f = 0.0, fscl;
1119        double          d = 0.0L, dscl;
 373        int             i;
 374        long long       l;
 375        float           f, fscl;
 376        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 = 1.552518092300708935e+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  *
```

```
 * This routine does not work if the instruction specified by *inst
 * is not a SIMD instruction.
 */
void
__fex_st_simd_result(ucontext_t *uap, sseinst_t *inst, enum fex_exception *e,
    fex_info_t *info)
{
        sseinst_t       dummy;
        int             i;

        /* store each part */
        switch (inst->op) {
        case cmpps:
                dummy.op = cmpss;
                dummy.imm = inst->imm;
                for (i = 0; i < 4; i++) {
                        dummy.op1 = (sseoperand_t *)&inst->op1->f[i];
                        dummy.op2 = (sseoperand_t *)&inst->op2->f[i];
                        __fex_st_sse_result(uap, &dummy, e[i], &info[i]);
                }
                break;

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

        case maxps:
                dummy.op = maxss;
                for (i = 0; i < 4; i++) {
                        dummy.op1 = (sseoperand_t *)&inst->op1->f[i];
                        dummy.op2 = (sseoperand_t *)&inst->op2->f[i];
                        __fex_st_sse_result(uap, &dummy, e[i], &info[i]);
                }
                break;

        case addps:
                dummy.op = addss;
                for (i = 0; i < 4; i++) {
                        dummy.op1 = (sseoperand_t *)&inst->op1->f[i];
                        dummy.op2 = (sseoperand_t *)&inst->op2->f[i];
                        __fex_st_sse_result(uap, &dummy, e[i], &info[i]);
                }
                break;

        case subps:
                dummy.op = subss;
                for (i = 0; i < 4; i++) {
                        dummy.op1 = (sseoperand_t *)&inst->op1->f[i];
                        dummy.op2 = (sseoperand_t *)&inst->op2->f[i];
                        __fex_st_sse_result(uap, &dummy, e[i], &info[i]);
                }
                break;

        case mulps:
                dummy.op = mulss;
                for (i = 0; i < 4; i++) {
                        dummy.op1 = (sseoperand_t *)&inst->op1->f[i];
                        dummy.op2 = (sseoperand_t *)&inst->op2->f[i];
                        __fex_st_sse_result(uap, &dummy, e[i], &info[i]);
                }
                break;
```



```
        case divps:
                dummy.op = divss;
                for (i = 0; i < 4; i++) {
                        dummy.op1 = (sseoperand_t *)&inst->op1->f[i];
                        dummy.op2 = (sseoperand_t *)&inst->op2->f[i];
                        __fex_st_sse_result(uap, &dummy, e[i], &info[i]);
                }
                break;

        case sqrtps:
                dummy.op = sqrtss;
                for (i = 0; i < 4; i++) {
                        dummy.op1 = (sseoperand_t *)&inst->op1->f[i];
                        dummy.op2 = (sseoperand_t *)&inst->op2->f[i];
                        __fex_st_sse_result(uap, &dummy, e[i], &info[i]);
                }
                break;

        case cvtdq2ps:
                dummy.op = cvtsi2ss;
                for (i = 0; i < 4; i++) {
                        dummy.op1 = (sseoperand_t *)&inst->op1->f[i];
                        dummy.op2 = (sseoperand_t *)&inst->op2->i[i];
                        __fex_st_sse_result(uap, &dummy, e[i], &info[i]);
                }
                break;

        case cvttps2dq:
                dummy.op = cvttss2si;
                for (i = 0; i < 4; i++) {
                        dummy.op1 = (sseoperand_t *)&inst->op1->i[i];
                        dummy.op2 = (sseoperand_t *)&inst->op2->f[i];
                        __fex_st_sse_result(uap, &dummy, e[i], &info[i]);
                }
                break;

        case cvtps2dq:
                dummy.op = cvtss2si;
                for (i = 0; i < 4; i++) {
                        dummy.op1 = (sseoperand_t *)&inst->op1->i[i];
                        dummy.op2 = (sseoperand_t *)&inst->op2->f[i];
                        __fex_st_sse_result(uap, &dummy, e[i], &info[i]);
                }
                break;

        case cvtpi2ps:
                dummy.op = cvtsi2ss;
                for (i = 0; i < 2; i++) {
                        dummy.op1 = (sseoperand_t *)&inst->op1->f[i];
                        dummy.op2 = (sseoperand_t *)&inst->op2->i[i];
                        __fex_st_sse_result(uap, &dummy, e[i], &info[i]);
                }
                break;

        case cvttps2pi:
                dummy.op = cvttss2si;
                for (i = 0; i < 2; i++) {
                        dummy.op1 = (sseoperand_t *)&inst->op1->i[i];
                        dummy.op2 = (sseoperand_t *)&inst->op2->f[i];
                        __fex_st_sse_result(uap, &dummy, e[i], &info[i]);
                }
                break;

        case cvtps2pi:
                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;

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;

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;

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;

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 */
```

```
**********************************************************
    1487 Sun May  4 03:06:20 2014
new/usr/src/lib/libm/common/m9x/fdim.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 fdim = __fdim
#endif

/*
 * fdim(x,y) returns x - y if x > y, +0 if x <= y, and NaN if x and
 * y are unordered.
 *
 * fdim(x,y) raises overflow or inexact if x > y and x - y overflows
 * or is inexact.  It raises invalid if either operand is a signaling
 * NaN.  Otherwise, it raises no exceptions.
 */

#include "libm.h"       /* for islessequal macro */

double
__fdim(double x, double y) {
#if defined(COMPARISON_MACRO_BUG)
        if (x == x && y == y && x <= y) {       /* } */
#else
        if (islessequal(x, y)) {
#endif
                x = 0.0;
                y = -x;
        }
        return (x - y);
}
```

*_____unchanged_portion_omitted_*

****
1549 Sun May 4 03:06:21 2014
new/usr/src/lib/libm/common/m9x/fdimf.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 fdimf = __fdimf
#endif

#include "libm.h"       /* for islessequal macro */

float
__fdimf(float x, float y) {
        /*
         * On SPARC v8plus/v9, this could be implemented as follows
         * (assuming %f0 = x, %f1 = y, return value left in %f0):
         *
         * fcmps        %fcc0,%f0,%f1
         * st           %g0,[scratch]   ! use fzero instead of st/ld
         * ld           [scratch],%f2   ! if VIS is available
         * fnegs        %f2,%f3
         * fmovsle      %fcc0,%f2,%f0
         * fmovsle      %fcc0,%f3,%f1
         * fsubs        %f0,%f1,%f0
         */
#if defined(COMPARISON_MACRO_BUG)
        if (x == x && y == y && x <= y) {       /* } */
#else
        if (islessequal(x, y)) {
#endif
                x = 0.0f;
                y = -x;
        }
        return (x - y);
}
```

_____unchanged_portion_omitted_

```
**********************************************************
    1223 Sun May  4 03:06:23 2014
new/usr/src/lib/libm/common/m9x/fdiml.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 fdiml = __fdiml
#endif

#include "libm.h"       /* for islessequal macro */

long double
__fdiml(long double x, long double y) {
#if defined(COMPARISON_MACRO_BUG)
        if (x == x && y == y && x <= y) {
#else
        if (islessequal(x, y)) {
#endif
                x = 0.0l;
                y = -x;
        }
        return (x - y);
}
______unchanged_portion_omitted_
```

```
**********************************************************
   12598 Sun May  4 03:06:24 2014
new/usr/src/lib/libm/common/m9x/fenv_inlines.h
**********************************************************
______unchanged_portion_omitted_

  42 extern __inline__ void
  43 __fenv_getcwsw(unsigned int *value)
  44 {
  45         union fp_cwsw *u = (union fp_cwsw *)value;
  45         union fp_cwsw ret;

  47         __asm__ __volatile__(
  48             "fstsw %0\n\t"
  49             "fstcw %1\n\t"
  50             : "=m" (u->words.cw), "=m" (u->words.sw));
  50             : "=m" (ret.words.cw), "=m" (ret.words.sw));
  51         *value = ret.cwsw;
  51 }

  53 extern __inline__ void
  54 __fenv_setcwsw(const unsigned int *value)
  55 {
  56         union fp_cwsw cwsw;
  57         short fenv[16];

  59         cwsw.cwsw = *value;

  61         __asm__ __volatile__(
  62             "fstenv %0\n\t"
  63             "movw   %4,%1\n\t"
  64             "movw   %3,%2\n\t"
  65             "fldenv %0\n\t"
  66             "fwait\n\t"
  67             : "=m" (fenv), "=m" (fenv[0]), "=m" (fenv[2])
  68             : "r" (cwsw.words.cw), "r" (cwsw.words.sw)
  69             : "d" (cwsw.words.cw), "c" (cwsw.words.sw)
  69             /* For practical purposes, we clobber the whole FPU */
  70             : "cc", "st", "st(1)", "st(2)", "st(3)", "st(4)", "st(5)",
  71               "st(6)", "st(7)");
  72 }

  74 extern __inline__ void
  75 __fenv_getmxcsr(unsigned int *value)
  76 {
  77         __asm__ __volatile__("stmxcsr %0" : "=m" (*value));
  78         __asm__ __volatile__("stmxcsr %1" : "+m" (*value));
  78 }
_____unchanged_portion_omitted_

  86 extern __inline__ long double
  87 f2xm1(long double x)
  88 {
  89         long double ret;

  91         __asm__ __volatile__("f2xm1" : "=t" (ret) : "0" (x) : "cc");
  92         __asm__ __volatile__("f2xm1" : "=t" (ret) : "0" (x));
  92         return (ret);
  93 }

  95 extern __inline__ long double
  96 fyl2x(long double y, long double x)
  97 {
  98         long double ret;

 100         __asm__ __volatile__("fyl2x"
```



```
 101             : "=t" (ret)
 102             : "0" (x), "u" (y)
 103             : "st(1)", "cc");
 101         __asm__ __volatile__("fyl2x" : "=t" (ret): "0" (x), "u" (y) : "st(1)");
 104         return (ret);
 105 }

 107 extern __inline__ long double
 108 fptan(long double x)
 109 {
 110         /*
 111          * fptan pushes 1.0 then the result on completion, so we want to pop
 112          * the FP stack twice, so we need a dummy value into which to pop it.
 113          */
 114         long double ret;
 115         long double dummy;

 117         __asm__ __volatile__("fptan"
 118             : "=t" (dummy), "=u" (ret)
 119             : "0" (x)
 120             : "cc");
 115         __asm__ __volatile__("fptan" : "=t" (dummy), "=u" (ret) : "0" (x));
 121         return (ret);
 122 }

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

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

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

 136         __asm__ __volatile__("fxtract" : "=t" (ret) : "0" (x));
 137         return (ret);
 141 }

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

 145         __asm__ __volatile__("fprem1" : "=t" (ret) : "0" (div), "u" (idend));
 146         return (ret);
 148 }

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

```
154         __asm__ __volatile__("fprem" : "=t" (ret) : "0" (div), "u" (idend));
155         return (ret);
155 }

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

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

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

175         __asm__ __volatile__("fsqrt" : "=t" (ret) : "0" (x));
176         return (ret);
174 }

176 extern __inline__ long double
177 fsincos(long double x)
178 {
179         __asm__ __volatile__("fsincos" : "+t" (x) : : "cc");
180         return (x);
182         long double ret;

184         __asm__ __volatile__("fsincos" : "=t" (ret) : "0" (x));
185         return (ret);
181 }

183 extern __inline__ long double
184 frndint(long double x)
185 {
186         __asm__ __volatile__("frndint" : "+t" (x) : : "cc");
187         return (x);
191         long double ret;

193         __asm__ __volatile__("frndint" : "=t" (ret) : "0" (x));
194         return (ret);
188 }

190 extern __inline__ long double
191 fscale(long double x, long double y)
192 {
193         long double ret;

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

199 extern __inline__ long double
200 fsin(long double x)
201 {
202         __asm__ __volatile__("fsin" : "+t" (x) : : "cc");
203         return (x);
```



```
209         long double ret;

211         __asm__ __volatile__("fsin" : "=t" (ret) : "0" (x));
212         return (ret);
204 }

206 extern __inline__ long double
207 fcos(long double x)
208 {
209         __asm__ __volatile__("fcos" : "+t" (x) : : "cc");
210         return (x);
218         long double ret;

220         __asm__ __volatile__("fcos" : "=t" (ret) : "0" (x));
221         return (ret);
211 }

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

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

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

246 extern __inline__ void
247 sse_cmpunordss(float *f1, float *f2, int *i1)
248 {
249         __asm__ __volatile__(
250             "cmpunordss %2, %1\n\t"
251             "movss      %1, %0"
252             : "=m" (*i1), "+x" (*f1)
253             : "x" (*f2)
254             : "cc");
260             : "=m" (*i1)
```

```
261             : "x" (*f1), "x" (*f2));
255 }

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

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

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

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

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

307 extern __inline__ void
308 sse_divss(float *f1, float *f2, float *f3)
309 {
```



```
310         __asm__ __volatile__(
311             "divss %2, %1\n\t"
312             "movss %1, %0"
313             : "=m" (*f3), "+x" (*f1)
314             : "x" (*f2));
320             : "=m" (*f3)
321             : "x" (*f1), "x" (*f2));
315 }

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

322 #endif /* ! codereview */
323         __asm__ __volatile__(
324             "sqrtss %2, %1\n\t"
325             "movss  %1, %0"
326             : "=m" (*f2), "=x" (tmp)
327             : "m" (*f1));
327             "sqrtss %1, %%xmm0\n\t"
328             "movss  %%xmm0, %0"
329             : "=m" (*f2)
330             : "m" (*f1)
331             : "xmm0");
328 }
```

_____***unchanged_portion_omitted_***

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

348 #endif /* ! codereview */
349         __asm__ __volatile__(
350             "cvtss2sd %2, %1\n\t"
351             "movsd    %1, %0"
352             : "=m" (*d1), "=x" (tmp)
353             : "m" (*f1));
350             "cvtss2sd %1, %%xmm0\n\t"
351             "movsd    %%xmm0, %0"
352             : "=m" (*d1)
353             : "m" (*f1)
354             : "xmm0");
354 }

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

361 #endif /* ! codereview */
362         __asm__ __volatile__(
363             "cvtsi2ss %2, %1\n\t"
364             "movss    %1, %0"
365             : "=m" (*f1), "=x" (tmp)
366             : "m" (*i1));
360             "cvtsi2ss %1, %%xmm0\n\t"
361             "movss    %%xmm0, %0"
362             : "=m" (*f1)
363             : "m" (*i1)
364             : "xmm0");
367 }

369 extern __inline__ void
370 sse_cvttss2si(float *f1, int *i1)
```

```
371 {
372         int tmp;

374 #endif /* ! codereview */
375         __asm__ __volatile__(
376             "cvttss2si %2, %1\n\t"
377             "movl      %1, %0"
378             : "=m" (*i1), "=r" (tmp)
379             : "m" (*f1));
370             "cvttss2si %1, %%ecx\n\t"
371             "movl      %%ecx, %0"
372             : "=m" (*i1)
373             : "m" (*f1)
374             : "ecx");
380 }

382 extern __inline__ void
383 sse_cvtss2si(float *f1, int *i1)
384 {
385         int tmp;

387 #endif /* ! codereview */
388         __asm__ __volatile__(
389             "cvtss2si %2, %1\n\t"
390             "movl     %1, %0"
391             : "=m" (*i1), "=r" (tmp)
392             : "m" (*f1));
380             "cvtss2si %1, %%ecx\n\t"
381             "movl     %%ecx, %0"
382             : "=m" (*i1)
383             : "m" (*f1)
384             : "ecx");
393 }

395 #if defined(__amd64)
396 extern __inline__ void
397 sse_cvtsi2ssq(long long *ll1, float *f1)
398 {
399         double tmp;

401 #endif /* ! codereview */
402         __asm__ __volatile__(
403             "cvtsi2ssq %2, %1\n\t"
404             "movss     %1, %0"
405             : "=m" (*f1), "=x" (tmp)
406             : "m" (*ll1));
391             "cvtsi2ssq %1, %%xmm0\n\t"
392             "movss     %%xmm0, %0"
393             : "=m" (*f1)
394             : "m" (*ll1)
395             : "xmm0");
407 }

409 extern __inline__ void
410 sse_cvttss2siq(float *f1, long long *ll1)
411 {
412         uint64_t tmp;

414 #endif /* ! codereview */
415         __asm__ __volatile__(
416             "cvttss2siq %2, %1\n\t"
417             "movq       %1, %0"
418             : "=m" (*ll1), "=r" (tmp)
419             : "m" (*f1));
401             "cvttss2siq %1, %%rcx\n\t"
402             "movq       %%rcx, %0"
```



```
403             : "=m" (*ll1)
404             : "m" (*f1)
405             : "rcx");
420 }

422 extern __inline__ void
423 sse_cvtss2siq(float *f1, long long *ll1)
424 {
425         uint64_t tmp;

427 #endif /* ! codereview */
428         __asm__ __volatile__(
429             "cvtss2siq %2, %1\n\t"
430             "movq      %1, %0"
431             : "=m" (*ll1), "=r" (tmp)
432             : "m" (*f1));
411             "cvtss2siq %1, %%rcx\n\t"
412             "movq      %%rcx, %0"
413             : "=m" (*ll1)
414             : "m" (*f1)
415             : "rcx");
433 }

435 #endif

437 extern __inline__ void
438 sse_cmpeqsd(double *d1, double *d2, long long *ll1)
439 {
440         __asm__ __volatile__(
441             "cmpeqsd %2,%1\n\t"
442             "movsd   %1,%0"
443             : "=m" (*ll1), "=x" (*d1)
444             : "x" (*d2));
426             : "=m" (*ll1)
427             : "x" (*d1), "x" (*d2));
445 }

447 extern __inline__ void
448 sse_cmpltsd(double *d1, double *d2, long long *ll1)
449 {
450         __asm__ __volatile__(
451             "cmpltsd %2,%1\n\t"
452             "movsd   %1,%0"
453             : "=m" (*ll1), "=x" (*d1)
454             : "x" (*d2));
436             : "=m" (*ll1)
437             : "x" (*d1), "x" (*d2));
455 }

457 extern __inline__ void
458 sse_cmplesd(double *d1, double *d2, long long *ll1)
459 {
460         __asm__ __volatile__(
461             "cmplesd %2,%1\n\t"
462             "movsd   %1,%0"
463             : "=m" (*ll1), "=x" (*d1)
464             : "x" (*d2));
446             : "=m" (*ll1)
447             : "x" (*d1), "x" (*d2));
465 }

467 extern __inline__ void
468 sse_cmpunordsd(double *d1, double *d2, long long *ll1)
469 {
470         __asm__ __volatile__(
471             "cmpunordsd %2,%1\n\t"
```

```
472             "movsd      %1,%0"
473             : "=m" (*ll1), "=x" (*d1)
474             : "x" (*d2));
456             : "=m" (*ll1)
457             : "x" (*d1), "x" (*d2));
475 }


478 extern __inline__ void
479 sse_minsd(double *d1, double *d2, double *d3)
480 {
481         __asm__ __volatile__(
482             "minsd %2,%1\n\t"
483             "movsd %1,%0"
484             : "=m" (*d3), "=x" (*d1)
485             : "x" (*d2));
467             : "=m" (*d3)
468             : "x" (*d1), "x" (*d2));
486 }

488 extern __inline__ void
489 sse_maxsd(double *d1, double *d2, double *d3)
490 {
491         __asm__ __volatile__(
492             "maxsd %2,%1\n\t"
493             "movsd %1,%0"
494             : "=m" (*d3), "=x" (*d1)
495             : "x" (*d2));
477             : "=m" (*d3)
478             : "x" (*d1), "x" (*d2));
496 }

498 extern __inline__ void
499 sse_addsd(double *d1, double *d2, double *d3)
500 {
501         __asm__ __volatile__(
502             "addsd %2,%1\n\t"
503             "movsd %1,%0"
504             : "=m" (*d3), "=x" (*d1)
505             : "x" (*d2));
487             : "=m" (*d3)
488             : "x" (*d1), "x" (*d2));
506 }

508 extern __inline__ void
509 sse_subsd(double *d1, double *d2, double *d3)
510 {
511         __asm__ __volatile__(
512             "subsd %2,%1\n\t"
513             "movsd %1,%0"
514             : "=m" (*d3), "=x" (*d1)
515             : "x" (*d2));
497             : "=m" (*d3)
498             : "x" (*d1), "x" (*d2));
516 }

518 extern __inline__ void
519 sse_mulsd(double *d1, double *d2, double *d3)
520 {
521         __asm__ __volatile__(
522             "mulsd %2,%1\n\t"
523             "movsd %1,%0"
524             : "=m" (*d3), "=x" (*d1)
525             : "x" (*d2));
507             : "=m" (*d3)
508             : "x" (*d1), "x" (*d2));
```



```
526 }

528 extern __inline__ void
529 sse_divsd(double *d1, double *d2, double *d3)
530 {
531         __asm__ __volatile__(
532             "divsd %2,%1\n\t"
533             "movsd %1,%0"
534             : "=m" (*d3), "=x" (*d1)
535             : "x" (*d2));
517             : "=m" (*d3)
518             : "x" (*d1), "x" (*d2)
519             : "xmm0");
536 }

538 extern __inline__ void
539 sse_sqrtsd(double *d1, double *d2)
540 {
541         double tmp;

543 #endif /* ! codereview */
544         __asm__ __volatile__(
545             "sqrtsd %2, %1\n\t"
546             "movsd %1, %0"
547             : "=m" (*d2), "=x" (tmp)
548             : "m" (*d1));
525             "sqrtsd %1, %%xmm0\n\t"
526             "movsd %%xmm0, %0"
527             : "=m" (*d2)
528             : "m" (*d1)
529             : "xmm0");
549 }
______unchanged_portion_omitted_

563 extern __inline__ void
564 sse_cvtsd2ss(double *d1, float *f1)
565 {
566         double tmp;

568 #endif /* ! codereview */
569         __asm__ __volatile__(
570             "cvtsd2ss %2,%1\n\t"
571             "movss    %1,%0"
572             : "=m" (*f1), "=x" (tmp)
573             : "m" (*d1));
547             "cvtsd2ss %1,%%xmm0\n\t"
548             "movss    %%xmm0,%0"
549             : "=m" (*f1)
550             : "m" (*d1)
551             : "xmm0");
574 }


576 extern __inline__ void
577 sse_cvtsi2sd(int *i1, double *d1)
578 {
579         double tmp;
580 #endif /* ! codereview */
581         __asm__ __volatile__(
582             "cvtsi2sd %2,%1\n\t"
583             "movsd    %1,%0"
584             : "=m" (*d1), "=x" (tmp)
585             : "m" (*i1));
558             "cvtsi2sd %1,%%xmm0\n\t"
559             "movsd    %%xmm0,%0"
560             : "=m" (*d1)
```

```
561             : "m" (*i1)
562             : "xmm0");
586 }

588 extern __inline__ void
589 sse_cvttsd2si(double *d1, int *i1)
590 {
591         int tmp;

593 #endif /* ! codereview */
594         __asm__ __volatile__(
595             "cvttsd2si %2,%1\n\t"
596             "movl      %1,%0"
597             : "=m" (*i1), "=r" (tmp)
598             : "m" (*d1));
568             "cvttsd2si %1,%%ecx\n\t"
569             "movl      %%ecx,%0"
570             : "=m" (*i1)
571             : "m" (*d1)
572             : "ecx");
599 }

601 extern __inline__ void
602 sse_cvtsd2si(double *d1, int *i1)
603 {
604         int tmp;

606 #endif /* ! codereview */
607         __asm__ __volatile__(
608             "cvtsd2si %2,%1\n\t"
609             "movl     %1,%0"
610             : "=m" (*i1), "=r" (tmp)
611             : "m" (*d1));
578             "cvtsd2si %1,%%ecx\n\t"
579             "movl     %%ecx,%0"
580             : "=m" (*i1)
581             : "m" (*d1)
582             : "ecx");
612 }

614 #if defined(__amd64)
615 extern __inline__ void
616 sse_cvtsi2sdq(long long *ll1, double *d1)
617 {
618         double tmp;

620 #endif /* ! codereview */
621         __asm__ __volatile__(
622             "cvtsi2sdq %2,%1\n\t"
623             "movsd     %1,%0"
624             : "=m" (*d1), "=x" (tmp)
625             : "m" (*ll1));
589             "cvtsi2sdq %1,%%xmm0\n\t"
590             "movsd     %%xmm0,%0"
591             : "=m" (*d1)
592             : "m" (*ll1)
593             : "xmm0");
626 }

628 extern __inline__ void
629 sse_cvttsd2siq(double *d1, long long *ll1)
630 {
631         uint64_t tmp;

633 #endif /* ! codereview */
634         __asm__ __volatile__(
```



```
635             "cvttsd2siq %2,%1\n\t"
636             "movq       %1,%0"
637             : "=m" (*ll1), "=r" (tmp)
638             : "m" (*d1));
599             "cvttsd2siq %1,%%rcx\n\t"
600             "movq       %%rcx,%0"
601             : "=m" (*ll1)
602             : "m" (*d1)
603             : "rcx");
639 }

641 extern __inline__ void
642 sse_cvtsd2siq(double *d1, long long *ll1)
643 {
644         uint64_t tmp;

646 #endif /* ! codereview */
647         __asm__ __volatile__(
648             "cvtsd2siq %2,%1\n\t"
649             "movq      %1,%0"
650             : "=m" (*ll1), "=r" (tmp)
651             : "m" (*d1));
609             "cvtsd2siq %1,%%rcx\n\t"
610             "movq      %%rcx,%0"
611             : "=m" (*ll1)
612             : "m" (*d1)
613             : "rcx");
652 }
653 #endif

655 #endif /* ! codereview */
656 #elif defined(__sparc)
657 extern __inline__ void
658 __fenv_getfsr(unsigned long *l)
659 {
660         __asm__ __volatile__(
661 #if defined(__sparcv9)
662                 "stx %%fsr,%0\n\t"
663 #else
664                 "st  %%fsr,%0\n\t"
665 #endif
666                 : "=m" (*l));
667 }

669 extern __inline__ void
670 __fenv_setfsr(const unsigned long *l)
671 {
672         __asm__ __volatile__(
673 #if defined(__sparcv9)
674                 "ldx %0,%%fsr\n\t"
675 #else
676                 "ld %0,%%fsr\n\t"
677 #endif
678                 : : "m" (*l) : "cc");
616         : : "m" (*l));
679 }
```

*_____unchanged_portion_omitted_*

```
**********************************************************
    6089 Sun May  4 03:06:27 2014
new/usr/src/lib/libm/common/m9x/fex_handler.h
**********************************************************
```

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

/* the following enums must match the bit positions in fenv.h */
enum fex_exception {
        fex_inexact             = 0,
        fex_division    = 1,
        fex_underflow   = 2,
        fex_overflow    = 3,
        fex_inv_zdz             = 4,
        fex_inv_idi             = 5,
        fex_inv_isi             = 6,
        fex_inv_zmi             = 7,
        fex_inv_sqrt    = 8,
        fex_inv_snan    = 9,
        fex_inv_int             = 10,
        fex_inv_cmp             = 11
};


/* auxiliary functions in __fex_hdlr.c */
extern struct fex_handler_data *__fex_get_thr_handlers(void);
extern void __fex_update_te(void);

/* auxiliary functions in __fex_sym.c */
extern void __fex_sym_init(void);
extern char *__fex_sym(char *, char **);

/* auxiliary functions in fex_log.c */
extern void __fex_mklog(ucontext_t *, char *, int, enum fex_exception,
        int, void *);

/* system-dependent auxiliary functions */
extern enum fex_exception __fex_get_invalid_type(siginfo_t *, ucontext_t *);
```



```
extern void __fex_get_op(siginfo_t *, ucontext_t *, fex_info_t *);
extern void __fex_st_result(siginfo_t *, ucontext_t *, fex_info_t *);

/* inline templates and macros for accessing fp state */
extern void __fenv_getfsr(unsigned long *);
extern void __fenv_setfsr(const unsigned long *);
extern void __fenv_setfsr(unsigned const long *);

#if defined(__sparc)

#define __fenv_get_rd(X)        ((X>>30)&0x3)
#define __fenv_set_rd(X,Y)      X=(X&~0xc0000000ul)|((Y)<<30)

#define __fenv_get_te(X)        ((X>>23)&0x1f)
#define __fenv_set_te(X,Y)      X=(X&~0x0f800000ul)|((Y)<<23)

#define __fenv_get_ex(X)        ((X>>5)&0x1f)
#define __fenv_set_ex(X,Y)      X=(X&~0x000003e0ul)|((Y)<<5)

#elif defined(__x86)

extern void __fenv_getcwsw(unsigned int *);
extern void __fenv_setcwsw(const unsigned int *);

extern void __fenv_getmxcsr(unsigned int *);
extern void __fenv_setmxcsr(const unsigned int *);

#define __fenv_get_rd(X)        ((X>>26)&3)
#define __fenv_set_rd(X,Y)      X=(X&~0x0c000000)|((Y)<<26)

#define __fenv_get_rp(X)        ((X>>24)&3)
#define __fenv_set_rp(X,Y)      X=(X&~0x03000000)|((Y)<<24)

#define __fenv_get_te(X)        ((X>>16)&0x3d)
#define __fenv_set_te(X,Y)      X=(X&~0x003d0000)|((Y)<<16)

#define __fenv_get_ex(X)        (X&0x3d)
#define __fenv_set_ex(X,Y)      X=(X&~0x0000003d)|(Y)

/*
 * These macros define some useful distinctions between various
 * SSE instructions.  In some cases, distinctions are made for
 * the purpose of simplifying the decoding of instructions, while
 * in other cases, they are made for the purpose of simplying the
 * emulation.  Note that these values serve as bit flags within
 * the enum values in sseinst_t.
 */
#define DOUBLE          0x100
#define SIMD            0x080
#define INTREG          0x040

typedef union {
        double          d[2];
        long long       l[2];
        float           f[4];
        int             i[4];
} sseoperand_t;
```

*unchanged_portion_omitted_*

****
9376 Sun May 4 03:06:29 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  4 03:06:31 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 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_***

```
496 #if 0
497 /*
498  * another fma for x86: assumes return value will be left in
499  * long double (80-bit double extended) precision
500  */
501 long double
502 __fma(double x, double y, double z) {
503         union {
504                 unsigned i[3];
505                 long double e;
506         } xx, yy, zz, tt;
507         long double xe, ye, xhi, xlo, yhi, ylo, zhi, zlo;
508         int ex, ey, ez;
509         unsigned cwsw, oldcwsw, s;

511         /* convert the operands to double extended */
512         xx.e = (long double) x;
513         yy.e = (long double) y;
514         zz.e = (long double) z;

516         /* extract the exponents of the arguments */
517         ex = xx.i[2] & 0x7fff;
518         ey = yy.i[2] & 0x7fff;
519         ez = zz.i[2] & 0x7fff;

521         /* dispense with inf, nan, and zero cases */
522         if (ex == 0x7fff || ey == 0x7fff || ex == 0 || ey == 0)
523                 /* x or y is inf, nan, or zero */
524                 return (xx.e * yy.e + zz.e);

526         if (ez >= 0x7fff) /* z is inf or nan */
527                 return (xx.e + zz.e);   /* avoid spurious inexact in x * y */

529         if (ez == 0) /* z is zero */
530                 return (xx.e * yy.e);   /* x * y isn't zero; no need to add z */

532         /*
533          * save the control and status words, mask all exceptions, and
534          * set rounding to 64-bit precision and to-nearest
535          */
536         __fenv_getcwsw(&oldcwsw);
537         cwsw = (oldcwsw & 0xf0c0ffff) | 0x033f0000;
538         __fenv_setcwsw(&cwsw);

540         /* multiply x*y to 106 bits */
541         xe = xx.e;
542         xx.i[0] = 0;
543         xhi = xx.e; /* hi 32 bits */
544         xlo = xe - xhi; /* lo 21 bits */
545         ye = yy.e;
546         yy.i[0] = 0;
```

```
547         yhi = yy.e;
548         ylo = ye - yhi;
549         xx.e = xe * ye;
550         xx.i[0] &= ~0x7ff; /* 53 bits of x*y */
551         yy.e = ((xhi * yhi - xx.e) + xhi * ylo + xlo * yhi) + xlo * ylo;

553         /* reduce to a sum of two terms */
554         if (yy.e != 0.0) {
555                 ex = xx.i[2] & 0x7fff;
556                 if (ez - ex > 10) {
557                         /* collapse y into a single bit and add to x */
558                         yy.i[0] = 0;
559                         yy.i[1] = 0x80000000;
560                         yy.i[2] = (yy.i[2] & 0x8000) | (ex - 60);
561                         xx.e += yy.e;
562                 } else if (ex - ez <= 10) {
563                         xx.e += zz.e; /* exact */
564                         zz.e = yy.e;
565                 } else if (ex - ez <= 42) {
566                         /* split z into two pieces */
567                         tt.i[0] = 0;
568                         tt.i[1] = 0x80000000;
569                         tt.i[2] = ex + 11;
570                         zhi = (zz.e + tt.e) - tt.e;
571                         zlo = zz.e - zhi;
572                         xx.e += zhi;
573                         zz.e = yy.e + zlo;
574                 } else if (ex - ez <= 63) {
575                         zz.e += yy.e; /* exact */
576                 } else if (ex - ez <= 106) {
577                         /*
578                          * collapse the tail of z into a sticky bit and add z
579                          * to y without error
580                          */
581                         if (ex - ez <= 81) {
582                                 s = 1 << (ex - ez - 50);
583                                 if (zz.i[0] & (s - 1))
584                                         zz.i[0] |= s;
585                                 zz.i[0] &= ~(s - 1);
586                         } else {
587                                 s = 1 << (ex - ez - 82);
588                                 if ((zz.i[1] & (s - 1)) | zz.i[0])
589                                         zz.i[1] |= s;
590                                 zz.i[1] &= ~(s - 1);
591                                 zz.i[0] = 0;
592                         }
593                         zz.e += yy.e;
594                 } else {
595                         /* collapse z into a single bit and add to y */
596                         zz.i[0] = 0;
597                         zz.i[1] = 0x80000000;
598                         zz.i[2] = (zz.i[2] & 0x8000) | (ex - 113);
599                         zz.e += yy.e;
600                 }
601         }

603         /* restore the control and status words, and sum */
604         __fenv_setcwsw(&oldcwsw);
605         return (xx.e + zz.e);
606 }
607 #endif
```

```
495 #else
496 #error Unknown architecture
497 #endif
```

****
28135 Sun May 4 03:06:33 2014
new/usr/src/lib/libm/common/m9x/fmal.c
****
_____unchanged_portion_omitted_

```
 62 #define half    C[0].d
 63 #define two     C[1].d
 64 #define twom16  C[2].d
 65 #define twom24  C[3].d
 66 #define two20   C[4].d
 67 #define twom28  C[5].d
 68 #define twom76  C[6].d
 69 #define twom124 C[7].d
 70 #define two36   C[8].d
 71 #define twom32  C[9].d
 72 #define huge    C[10].d
 73 #define tiny    C[11].d
 74 #define tiny2   C[12].d
 75 #define zero    C[13].d
 76 #define inf     C[14].d
 77 #define snan    C[15].d

 79 static const unsigned int fsr_rm = 0xc0000000u;

 81 /*
 82  * fmal for SPARC: 128-bit quad precision, big-endian
 83  */
 84 long double
 85 __fmal(long double x, long double y, long double z) {
 86         union {
 87                 unsigned int i[4];
 87                 unsigned i[4];
 88                 long double q;
 89         } xx, yy, zz;
 90         union {
 91                 unsigned int i[2];
 91                 unsigned i[2];
 92                 double d;
 93         } u;
 94         double dx[5], dy[5], dxy[9], c, s;
 95         unsigned int xy0, xy1, xy2, xy3, xy4, xy5, xy6, xy7;
 96         unsigned int z0, z1, z2, z3, z4, z5, z6, z7;
 97         unsigned int rm, sticky;
 95         unsigned xy0, xy1, xy2, xy3, xy4, xy5, xy6, xy7;
 96         unsigned z0, z1, z2, z3, z4, z5, z6, z7;
 97         unsigned rm, sticky;
 98         unsigned int fsr;
 99         int hx, hy, hz, ex, ey, ez, exy, sxy, sz, e, ibit;
100         int cx, cy, cz;
101         volatile double dummy;

103         /* extract the high order words of the arguments */
104         xx.q = x;
105         yy.q = y;
106         zz.q = z;
107         hx = xx.i[0] & ~0x80000000;
108         hy = yy.i[0] & ~0x80000000;
109         hz = zz.i[0] & ~0x80000000;

111         /*
112          * distinguish zero, finite nonzero, infinite, and quiet nan
113          * arguments; raise invalid and return for signaling nans
114          */
115         if (hx >= 0x7fff0000) {
116                 if ((hx & 0xffff) | xx.i[1] | xx.i[2] | xx.i[3]) {
```



```
117                         if (!(hx & 0x8000)) {
118                                 /* signaling nan, raise invalid */
119                                 dummy = snan;
120                                 dummy += snan;
121                                 xx.i[0] |= 0x8000;
122                                 return (xx.q);
123                         }
124                         cx = 3; /* quiet nan */
125                 } else
126                         cx = 2; /* inf */
127         } else if (hx == 0) {
128                 cx = (xx.i[1] | xx.i[2] | xx.i[3]) ? 1 : 0;
129                                 /* subnormal or zero */
130         } else
131                 cx = 1;         /* finite nonzero */

133         if (hy >= 0x7fff0000) {
134                 if ((hy & 0xffff) | yy.i[1] | yy.i[2] | yy.i[3]) {
135                         if (!(hy & 0x8000)) {
136                                 dummy = snan;
137                                 dummy += snan;
138                                 yy.i[0] |= 0x8000;
139                                 return (yy.q);
140                         }
141                         cy = 3;
142                 } else
143                         cy = 2;
144         } else if (hy == 0) {
145                 cy = (yy.i[1] | yy.i[2] | yy.i[3]) ? 1 : 0;
146         } else
147                 cy = 1;

149         if (hz >= 0x7fff0000) {
150                 if ((hz & 0xffff) | zz.i[1] | zz.i[2] | zz.i[3]) {
151                         if (!(hz & 0x8000)) {
152                                 dummy = snan;
153                                 dummy += snan;
154                                 zz.i[0] |= 0x8000;
155                                 return (zz.q);
156                         }
157                         cz = 3;
158                 } else
159                         cz = 2;
160         } else if (hz == 0) {
161                 cz = (zz.i[1] | zz.i[2] | zz.i[3]) ? 1 : 0;
162         } else
163                 cz = 1;

165         /* get the fsr and clear current exceptions */
166         __fenv_getfsr32(&fsr);
167         fsr &= ~FSR_CEXC;

169         /* handle all other zero, inf, and nan cases */
170         if (cx != 1 || cy != 1 || cz != 1) {
171                 /* if x or y is a quiet nan, return it */
172                 if (cx == 3) {
173                         __fenv_setfsr32(&fsr);
174                         return (x);
175                 }
176                 if (cy == 3) {
177                         __fenv_setfsr32(&fsr);
178                         return (y);
179                 }

181                 /* if x*y is 0*inf, raise invalid and return the default nan */
182                 if ((cx == 0 && cy == 2) || (cx == 2 && cy == 0)) {
```

```
183                         dummy = zero;
184                         dummy *= inf;
185                         zz.i[0] = 0x7fffffff;
186                         zz.i[1] = zz.i[2] = zz.i[3] = 0xffffffff;
187                         return (zz.q);
188                 }

190                 /* if z is a quiet nan, return it */
191                 if (cz == 3) {
192                         __fenv_setfsr32(&fsr);
193                         return (z);
194                 }

196                 /*
197                  * now none of x, y, or z is nan; handle cases where x or y
198                  * is inf
199                  */
200                 if (cx == 2 || cy == 2) {
201                         /*
202                          * if z is also inf, either we have inf-inf or
203                          * the result is the same as z depending on signs
204                          */
205                         if (cz == 2) {
206                                 if ((int) ((xx.i[0] ^ yy.i[0]) ^ zz.i[0]) < 0) {
207                                         dummy = inf;
208                                         dummy -= inf;
209                                         zz.i[0] = 0x7fffffff;
210                                         zz.i[1] = zz.i[2] = zz.i[3] =
211                                             0xffffffff;
212                                         return (zz.q);
213                                 }
214                                 __fenv_setfsr32(&fsr);
215                                 return (z);
216                         }

218                         /* otherwise the result is inf with appropriate sign */
219                         zz.i[0] = ((xx.i[0] ^ yy.i[0]) & 0x80000000) |
220                             0x7fff0000;
221                         zz.i[1] = zz.i[2] = zz.i[3] = 0;
222                         __fenv_setfsr32(&fsr);
223                         return (zz.q);
224                 }

226                 /* if z is inf, return it */
227                 if (cz == 2) {
228                         __fenv_setfsr32(&fsr);
229                         return (z);
230                 }

232                 /*
233                  * now x, y, and z are all finite; handle cases where x or y
234                  * is zero
235                  */
236                 if (cx == 0 || cy == 0) {
237                         /* either we have 0-0 or the result is the same as z */
238                         if (cz == 0 && (int) ((xx.i[0] ^ yy.i[0]) ^ zz.i[0]) <
239                                 0) {
240                                 zz.i[0] = (fsr >> 30) == FSR_RM ? 0x80000000 :
241                                         0;
242                                 __fenv_setfsr32(&fsr);
243                                 return (zz.q);
244                         }
245                         __fenv_setfsr32(&fsr);
246                         return (z);
247                 }
```



```
249                 /* if we get here, x and y are nonzero finite, z must be zero */
250                 return (x * y);
251         }

253         /*
254          * now x, y, and z are all finite and nonzero; set round-to-
255          * negative-infinity mode
256          */
257         __fenv_setfsr32(&fsr_rm);

259         /*
260          * get the signs and exponents and normalize the significands
261          * of x and y
262          */
263         sxy = (xx.i[0] ^ yy.i[0]) & 0x80000000;
264         ex = hx >> 16;
265         hx &= 0xffff;
266         if (!ex) {
267                 if (hx | (xx.i[1] & 0xfffe0000)) {
268                         ex = 1;
269                 } else if (xx.i[1] | (xx.i[2] & 0xfffe0000)) {
270                         hx = xx.i[1];
271                         xx.i[1] = xx.i[2];
272                         xx.i[2] = xx.i[3];
273                         xx.i[3] = 0;
274                         ex = -31;
275                 } else if (xx.i[2] | (xx.i[3] & 0xfffe0000)) {
276                         hx = xx.i[2];
277                         xx.i[1] = xx.i[3];
278                         xx.i[2] = xx.i[3] = 0;
279                         ex = -63;
280                 } else {
281                         hx = xx.i[3];
282                         xx.i[1] = xx.i[2] = xx.i[3] = 0;
283                         ex = -95;
284                 }
285                 while ((hx & 0x10000) == 0) {
286                         hx = (hx << 1) | (xx.i[1] >> 31);
287                         xx.i[1] = (xx.i[1] << 1) | (xx.i[2] >> 31);
288                         xx.i[2] = (xx.i[2] << 1) | (xx.i[3] >> 31);
289                         xx.i[3] <<= 1;
290                         ex--;
291                 }
292         } else
293                 hx |= 0x10000;
294         ey = hy >> 16;
295         hy &= 0xffff;
296         if (!ey) {
297                 if (hy | (yy.i[1] & 0xfffe0000)) {
298                         ey = 1;
299                 } else if (yy.i[1] | (yy.i[2] & 0xfffe0000)) {
300                         hy = yy.i[1];
301                         yy.i[1] = yy.i[2];
302                         yy.i[2] = yy.i[3];
303                         yy.i[3] = 0;
304                         ey = -31;
305                 } else if (yy.i[2] | (yy.i[3] & 0xfffe0000)) {
306                         hy = yy.i[2];
307                         yy.i[1] = yy.i[3];
308                         yy.i[2] = yy.i[3] = 0;
309                         ey = -63;
310                 } else {
311                         hy = yy.i[3];
312                         yy.i[1] = yy.i[2] = yy.i[3] = 0;
313                         ey = -95;
314                 }
```

```
315                 while ((hy & 0x10000) == 0) {
316                         hy = (hy << 1) | (yy.i[1] >> 31);
317                         yy.i[1] = (yy.i[1] << 1) | (yy.i[2] >> 31);
318                         yy.i[2] = (yy.i[2] << 1) | (yy.i[3] >> 31);
319                         yy.i[3] <<= 1;
320                         ey--;
321                 }
322         } else
323                 hy |= 0x10000;
324         exy = ex + ey - 0x3fff;
325
326         /* convert the significands of x and y to doubles */
327         c = twom16;
328         dx[0] = (double) ((int) hx) * c;
329         dy[0] = (double) ((int) hy) * c;
330
331         c *= twom24;
332         dx[1] = (double) ((int) (xx.i[1] >> 8)) * c;
333         dy[1] = (double) ((int) (yy.i[1] >> 8)) * c;
334
335         c *= twom24;
336         dx[2] = (double) ((int) (((xx.i[1] << 16) | (xx.i[2] >> 16)) &
337             0xffffff)) * c;
338         dy[2] = (double) ((int) (((yy.i[1] << 16) | (yy.i[2] >> 16)) &
339             0xffffff)) * c;
340
341         c *= twom24;
342         dx[3] = (double) ((int) (((xx.i[2] << 8) | (xx.i[3] >> 24)) &
343             0xffffff)) * c;
344         dy[3] = (double) ((int) (((yy.i[2] << 8) | (yy.i[3] >> 24)) &
345             0xffffff)) * c;
346
347         c *= twom24;
348         dx[4] = (double) ((int) (xx.i[3] & 0xffffff)) * c;
349         dy[4] = (double) ((int) (yy.i[3] & 0xffffff)) * c;
350
351         /* form the "digits" of the product */
352         dxy[0] = dx[0] * dy[0];
353         dxy[1] = dx[0] * dy[1] + dx[1] * dy[0];
354         dxy[2] = dx[0] * dy[2] + dx[1] * dy[1] + dx[2] * dy[0];
355         dxy[3] = dx[0] * dy[3] + dx[1] * dy[2] + dx[2] * dy[1] +
356             dx[3] * dy[0];
357         dxy[4] = dx[0] * dy[4] + dx[1] * dy[3] + dx[2] * dy[2] +
358             dx[3] * dy[1] + dx[4] * dy[0];
359         dxy[5] = dx[1] * dy[4] + dx[2] * dy[3] + dx[3] * dy[2] +
360             dx[4] * dy[1];
361         dxy[6] = dx[2] * dy[4] + dx[3] * dy[3] + dx[4] * dy[2];
362         dxy[7] = dx[3] * dy[4] + dx[4] * dy[3];
363         dxy[8] = dx[4] * dy[4];
364
365         /* split odd-numbered terms and combine into even-numbered terms */
366         c = (dxy[1] + two20) - two20;
367         dxy[0] += c;
368         dxy[1] -= c;
369         c = (dxy[3] + twom28) - twom28;
370         dxy[2] += c + dxy[1];
371         dxy[3] -= c;
372         c = (dxy[5] + twom76) - twom76;
373         dxy[4] += c + dxy[3];
374         dxy[5] -= c;
375         c = (dxy[7] + twom124) - twom124;
376         dxy[6] += c + dxy[5];
377         dxy[8] += (dxy[7] - c);
378
379         /* propagate carries, adjusting the exponent if need be */
380         dxy[7] = dxy[6] + dxy[8];
```



```
381         dxy[5] = dxy[4] + dxy[7];
382         dxy[3] = dxy[2] + dxy[5];
383         dxy[1] = dxy[0] + dxy[3];
384         if (dxy[1] >= two) {
385                 dxy[0] *= half;
386                 dxy[1] *= half;
387                 dxy[2] *= half;
388                 dxy[3] *= half;
389                 dxy[4] *= half;
390                 dxy[5] *= half;
391                 dxy[6] *= half;
392                 dxy[7] *= half;
393                 dxy[8] *= half;
394                 exy++;
395         }
396
397         /* extract the significand of x*y */
398         s = two36;
399         u.d = c = dxy[1] + s;
400         xy0 = u.i[1];
401         c -= s;
402         dxy[1] -= c;
403         dxy[0] -= c;
404
405         s *= twom32;
406         u.d = c = dxy[1] + s;
407         xy1 = u.i[1];
408         c -= s;
409         dxy[2] += (dxy[0] - c);
410         dxy[3] = dxy[2] + dxy[5];
411
412         s *= twom32;
413         u.d = c = dxy[3] + s;
414         xy2 = u.i[1];
415         c -= s;
416         dxy[4] += (dxy[2] - c);
417         dxy[5] = dxy[4] + dxy[7];
418
419         s *= twom32;
420         u.d = c = dxy[5] + s;
421         xy3 = u.i[1];
422         c -= s;
423         dxy[4] -= c;
424         dxy[5] = dxy[4] + dxy[7];
425
426         s *= twom32;
427         u.d = c = dxy[5] + s;
428         xy4 = u.i[1];
429         c -= s;
430         dxy[6] += (dxy[4] - c);
431         dxy[7] = dxy[6] + dxy[8];
432
433         s *= twom32;
434         u.d = c = dxy[7] + s;
435         xy5 = u.i[1];
436         c -= s;
437         dxy[8] += (dxy[6] - c);
438
439         s *= twom32;
440         u.d = c = dxy[8] + s;
441         xy6 = u.i[1];
442         c -= s;
443         dxy[8] -= c;
444
445         s *= twom32;
446         u.d = c = dxy[8] + s;
```

```
447         xy7 = u.i[1];
448
449         /* extract the sign, exponent, and significand of z */
450         sz = zz.i[0] & 0x80000000;
451         ez = hz >> 16;
452         z0 = hz & 0xffff;
453         if (!ez) {
454                 if (z0 | (zz.i[1] & 0xfffe0000)) {
455                         z1 = zz.i[1];
456                         z2 = zz.i[2];
457                         z3 = zz.i[3];
458                         ez = 1;
459                 } else if (zz.i[1] | (zz.i[2] & 0xfffe0000)) {
460                         z0 = zz.i[1];
461                         z1 = zz.i[2];
462                         z2 = zz.i[3];
463                         z3 = 0;
464                         ez = -31;
465                 } else if (zz.i[2] | (zz.i[3] & 0xfffe0000)) {
466                         z0 = zz.i[2];
467                         z1 = zz.i[3];
468                         z2 = z3 = 0;
469                         ez = -63;
470                 } else {
471                         z0 = zz.i[3];
472                         z1 = z2 = z3 = 0;
473                         ez = -95;
474                 }
475                 while ((z0 & 0x10000) == 0) {
476                         z0 = (z0 << 1) | (z1 >> 31);
477                         z1 = (z1 << 1) | (z2 >> 31);
478                         z2 = (z2 << 1) | (z3 >> 31);
479                         z3 <<= 1;
480                         ez--;
481                 }
482         } else {
483                 z0 |= 0x10000;
484                 z1 = zz.i[1];
485                 z2 = zz.i[2];
486                 z3 = zz.i[3];
487         }
488         z4 = z5 = z6 = z7 = 0;
489
490         /*
491          * now x*y is represented by sxy, exy, and xy[0-7], and z is
492          * represented likewise; swap if need be so |xy| <= |z|
493          */
494         if (exy > ez || (exy == ez && (xy0 > z0 || (xy0 == z0 && (xy1 > z1 ||
495                 (xy1 == z1 && (xy2 > z2 || (xy2 == z2 && (xy3 > z3 ||
496                 (xy3 == z3 && (xy4 | xy5 | xy6 | xy7) != 0))))))))))) {
497                 e = sxy; sxy = sz; sz = e;
498                 e = exy; exy = ez; ez = e;
499                 e = xy0; xy0 = z0; z0 = e;
500                 e = xy1; xy1 = z1; z1 = e;
501                 e = xy2; xy2 = z2; z2 = e;
502                 e = xy3; xy3 = z3; z3 = e;
503                 z4 = xy4; xy4 = 0;
504                 z5 = xy5; xy5 = 0;
505                 z6 = xy6; xy6 = 0;
506                 z7 = xy7; xy7 = 0;
507         }
508
509         /* shift the significand of xy keeping a sticky bit */
510         e = ez - exy;
511         if (e > 236) {
512                 xy0 = xy1 = xy2 = xy3 = xy4 = xy5 = xy6 = 0;
```



```
513                 xy7 = 1;
514         } else if (e >= 224) {
515                 sticky = xy7 | xy6 | xy5 | xy4 | xy3 | xy2 | xy1 |
516                         ((xy0 << 1) << (255 - e));
517                 xy7 = xy0 >> (e - 224);
518                 if (sticky)
519                         xy7 |= 1;
520                 xy0 = xy1 = xy2 = xy3 = xy4 = xy5 = xy6 = 0;
521         } else if (e >= 192) {
522                 sticky = xy7 | xy6 | xy5 | xy4 | xy3 | xy2 |
523                         ((xy1 << 1) << (223 - e));
524                 xy7 = (xy1 >> (e - 192)) | ((xy0 << 1) << (223 - e));
525                 if (sticky)
526                         xy7 |= 1;
527                 xy6 = xy0 >> (e - 192);
528                 xy0 = xy1 = xy2 = xy3 = xy4 = xy5 = 0;
529         } else if (e >= 160) {
530                 sticky = xy7 | xy6 | xy5 | xy4 | xy3 |
531                         ((xy2 << 1) << (191 - e));
532                 xy7 = (xy2 >> (e - 160)) | ((xy1 << 1) << (191 - e));
533                 if (sticky)
534                         xy7 |= 1;
535                 xy6 = (xy1 >> (e - 160)) | ((xy0 << 1) << (191 - e));
536                 xy5 = xy0 >> (e - 160);
537                 xy0 = xy1 = xy2 = xy3 = xy4 = 0;
538         } else if (e >= 128) {
539                 sticky = xy7 | xy6 | xy5 | xy4 | ((xy3 << 1) << (159 - e));
540                 xy7 = (xy3 >> (e - 128)) | ((xy2 << 1) << (159 - e));
541                 if (sticky)
542                         xy7 |= 1;
543                 xy6 = (xy2 >> (e - 128)) | ((xy1 << 1) << (159 - e));
544                 xy5 = (xy1 >> (e - 128)) | ((xy0 << 1) << (159 - e));
545                 xy4 = xy0 >> (e - 128);
546                 xy0 = xy1 = xy2 = xy3 = 0;
547         } else if (e >= 96) {
548                 sticky = xy7 | xy6 | xy5 | ((xy4 << 1) << (127 - e));
549                 xy7 = (xy4 >> (e - 96)) | ((xy3 << 1) << (127 - e));
550                 if (sticky)
551                         xy7 |= 1;
552                 xy6 = (xy3 >> (e - 96)) | ((xy2 << 1) << (127 - e));
553                 xy5 = (xy2 >> (e - 96)) | ((xy1 << 1) << (127 - e));
554                 xy4 = (xy1 >> (e - 96)) | ((xy0 << 1) << (127 - e));
555                 xy3 = xy0 >> (e - 96);
556                 xy0 = xy1 = xy2 = 0;
557         } else if (e >= 64) {
558                 sticky = xy7 | xy6 | ((xy5 << 1) << (95 - e));
559                 xy7 = (xy5 >> (e - 64)) | ((xy4 << 1) << (95 - e));
560                 if (sticky)
561                         xy7 |= 1;
562                 xy6 = (xy4 >> (e - 64)) | ((xy3 << 1) << (95 - e));
563                 xy5 = (xy3 >> (e - 64)) | ((xy2 << 1) << (95 - e));
564                 xy4 = (xy2 >> (e - 64)) | ((xy1 << 1) << (95 - e));
565                 xy3 = (xy1 >> (e - 64)) | ((xy0 << 1) << (95 - e));
566                 xy2 = xy0 >> (e - 64);
567                 xy0 = xy1 = 0;
568         } else if (e >= 32) {
569                 sticky = xy7 | ((xy6 << 1) << (63 - e));
570                 xy7 = (xy6 >> (e - 32)) | ((xy5 << 1) << (63 - e));
571                 if (sticky)
572                         xy7 |= 1;
573                 xy6 = (xy5 >> (e - 32)) | ((xy4 << 1) << (63 - e));
574                 xy5 = (xy4 >> (e - 32)) | ((xy3 << 1) << (63 - e));
575                 xy4 = (xy3 >> (e - 32)) | ((xy2 << 1) << (63 - e));
576                 xy3 = (xy2 >> (e - 32)) | ((xy1 << 1) << (63 - e));
577                 xy2 = (xy1 >> (e - 32)) | ((xy0 << 1) << (63 - e));
578                 xy1 = xy0 >> (e - 32);
```

```
579                 xy0 = 0;
580         } else if (e) {
581                 sticky = (xy7 << 1) << (31 - e);
582                 xy7 = (xy7 >> e) | ((xy6 << 1) << (31 - e));
583                 if (sticky)
584                         xy7 |= 1;
585                 xy6 = (xy6 >> e) | ((xy5 << 1) << (31 - e));
586                 xy5 = (xy5 >> e) | ((xy4 << 1) << (31 - e));
587                 xy4 = (xy4 >> e) | ((xy3 << 1) << (31 - e));
588                 xy3 = (xy3 >> e) | ((xy2 << 1) << (31 - e));
589                 xy2 = (xy2 >> e) | ((xy1 << 1) << (31 - e));
590                 xy1 = (xy1 >> e) | ((xy0 << 1) << (31 - e));
591                 xy0 >>= e;
592         }

594         /* if this is a magnitude subtract, negate the significand of xy */
595         if (sxy ^ sz) {
596                 xy0 = ~xy0;
597                 xy1 = ~xy1;
598                 xy2 = ~xy2;
599                 xy3 = ~xy3;
600                 xy4 = ~xy4;
601                 xy5 = ~xy5;
602                 xy6 = ~xy6;
603                 xy7 = -xy7;
604                 if (xy7 == 0)
605                         if (++xy6 == 0)
606                                 if (++xy5 == 0)
607                                         if (++xy4 == 0)
608                                                 if (++xy3 == 0)
609                                                         if (++xy2 == 0)
610                                                                 if (++xy1 == 0)
611                                                                         xy0++;
612         }

614         /* add, propagating carries */
615         z7 += xy7;
616         e = (z7 < xy7);
617         z6 += xy6;
618         if (e) {
619                 z6++;
620                 e = (z6 <= xy6);
621         } else
622                 e = (z6 < xy6);
623         z5 += xy5;
624         if (e) {
625                 z5++;
626                 e = (z5 <= xy5);
627         } else
628                 e = (z5 < xy5);
629         z4 += xy4;
630         if (e) {
631                 z4++;
632                 e = (z4 <= xy4);
633         } else
634                 e = (z4 < xy4);
635         z3 += xy3;
636         if (e) {
637                 z3++;
638                 e = (z3 <= xy3);
639         } else
640                 e = (z3 < xy3);
641         z2 += xy2;
642         if (e) {
643                 z2++;
644                 e = (z2 <= xy2);
```



```
645         } else
646                 e = (z2 < xy2);
647         z1 += xy1;
648         if (e) {
649                 z1++;
650                 e = (z1 <= xy1);
651         } else
652                 e = (z1 < xy1);
653         z0 += xy0;
654         if (e)
655                 z0++;

657         /* postnormalize and collect rounding information into z4 */
658         if (ez < 1) {
659                 /* result is tiny; shift right until exponent is within range */
660                 e = 1 - ez;
661                 if (e > 116) {
662                         z4 = 1; /* result can't be exactly zero */
663                         z0 = z1 = z2 = z3 = 0;
664                 } else if (e >= 96) {
665                         sticky = z7 | z6 | z5 | z4 | z3 | z2 |
666                                 ((z1 << 1) << (127 - e));
667                         z4 = (z1 >> (e - 96)) | ((z0 << 1) << (127 - e));
668                         if (sticky)
669                                 z4 |= 1;
670                         z3 = z0 >> (e - 96);
671                         z0 = z1 = z2 = 0;
672                 } else if (e >= 64) {
673                         sticky = z7 | z6 | z5 | z4 | z3 |
674                                 ((z2 << 1) << (95 - e));
675                         z4 = (z2 >> (e - 64)) | ((z1 << 1) << (95 - e));
676                         if (sticky)
677                                 z4 |= 1;
678                         z3 = (z1 >> (e - 64)) | ((z0 << 1) << (95 - e));
679                         z2 = z0 >> (e - 64);
680                         z0 = z1 = 0;
681                 } else if (e >= 32) {
682                         sticky = z7 | z6 | z5 | z4 | ((z3 << 1) << (63 - e));
683                         z4 = (z3 >> (e - 32)) | ((z2 << 1) << (63 - e));
684                         if (sticky)
685                                 z4 |= 1;
686                         z3 = (z2 >> (e - 32)) | ((z1 << 1) << (63 - e));
687                         z2 = (z1 >> (e - 32)) | ((z0 << 1) << (63 - e));
688                         z1 = z0 >> (e - 32);
689                         z0 = 0;
690                 } else {
691                         sticky = z7 | z6 | z5 | (z4 << 1) << (31 - e);
692                         z4 = (z4 >> e) | ((z3 << 1) << (31 - e));
693                         if (sticky)
694                                 z4 |= 1;
695                         z3 = (z3 >> e) | ((z2 << 1) << (31 - e));
696                         z2 = (z2 >> e) | ((z1 << 1) << (31 - e));
697                         z1 = (z1 >> e) | ((z0 << 1) << (31 - e));
698                         z0 >>= e;
699                 }
700                 ez = 1;
701         } else if (z0 >= 0x20000) {
702                 /* carry out; shift right by one */
703                 sticky = (z4 & 1) | z5 | z6 | z7;
704                 z4 = (z4 >> 1) | (z3 << 31);
705                 if (sticky)
706                         z4 |= 1;
707                 z3 = (z3 >> 1) | (z2 << 31);
708                 z2 = (z2 >> 1) | (z1 << 31);
709                 z1 = (z1 >> 1) | (z0 << 31);
710                 z0 >>= 1;
```

```
711                 ez++;
712         } else {
713                 if (z0 < 0x10000 && (z0 | z1 | z2 | z3 | z4 | z5 | z6 | z7)
714                         != 0) {
715                         /*
716                          * borrow/cancellation; shift left as much as
717                          * exponent allows
718                          */
719                         while (!(z0 | (z1 & 0xfffe0000)) && ez >= 33) {
720                                 z0 = z1;
721                                 z1 = z2;
722                                 z2 = z3;
723                                 z3 = z4;
724                                 z4 = z5;
725                                 z5 = z6;
726                                 z6 = z7;
727                                 z7 = 0;
728                                 ez -= 32;
729                         }
730                         while (z0 < 0x10000 && ez > 1) {
731                                 z0 = (z0 << 1) | (z1 >> 31);
732                                 z1 = (z1 << 1) | (z2 >> 31);
733                                 z2 = (z2 << 1) | (z3 >> 31);
734                                 z3 = (z3 << 1) | (z4 >> 31);
735                                 z4 = (z4 << 1) | (z5 >> 31);
736                                 z5 = (z5 << 1) | (z6 >> 31);
737                                 z6 = (z6 << 1) | (z7 >> 31);
738                                 z7 <<= 1;
739                                 ez--;
740                         }
741                 }
742                 if (z5 | z6 | z7)
743                         z4 |= 1;
744         }
745
746         /* get the rounding mode */
747         rm = fsr >> 30;
748
749         /* strip off the integer bit, if there is one */
750         ibit = z0 & 0x10000;
751         if (ibit)
752                 z0 -= 0x10000;
753         else {
754                 ez = 0;
755                 if (!(z0 | z1 | z2 | z3 | z4)) { /* exact zero */
756                         zz.i[0] = rm == FSR_RM ? 0x80000000 : 0;
757                         zz.i[1] = zz.i[2] = zz.i[3] = 0;
758                         __fenv_setfsr32(&fsr);
759                         return (zz.q);
760                 }
761         }
762
763         /*
764          * flip the sense of directed roundings if the result is negative;
765          * the logic below applies to a positive result
766          */
767         if (sz)
768                 rm ^= rm >> 1;
769
770         /* round and raise exceptions */
771         if (z4) {
772                 fsr |= FSR_NXC;
773
774                 /* decide whether to round the fraction up */
775                 if (rm == FSR_RP || (rm == FSR_RN && (z4 > 0x80000000u ||
776                         (z4 == 0x80000000u && (z3 & 1))))) {
```



```
777                         /* round up and renormalize if necessary */
778                         if (++z3 == 0)
779                                 if (++z2 == 0)
780                                         if (++z1 == 0)
781                                                 if (++z0 == 0x10000) {
782                                                         z0 = 0;
783                                                         ez++;
784                                                 }
785                 }
786         }
787
788         /* check for under/overflow */
789         if (ez >= 0x7fff) {
790                 if (rm == FSR_RN || rm == FSR_RP) {
791                         zz.i[0] = sz | 0x7fff0000;
792                         zz.i[1] = zz.i[2] = zz.i[3] = 0;
793                 } else {
794                         zz.i[0] = sz | 0x7fefffff;
795                         zz.i[1] = zz.i[2] = zz.i[3] = 0xffffffff;
796                 }
797                 fsr |= FSR_OFC | FSR_NXC;
798         } else {
799                 zz.i[0] = sz | (ez << 16) | z0;
800                 zz.i[1] = z1;
801                 zz.i[2] = z2;
802                 zz.i[3] = z3;
803
804                 /*
805                  * !ibit => exact result was tiny before rounding,
806                  * z4 nonzero => result delivered is inexact
807                  */
808                 if (!ibit) {
809                         if (z4)
810                                 fsr |= FSR_UFC | FSR_NXC;
811                         else if (fsr & FSR_UFM)
812                                 fsr |= FSR_UFC;
813                 }
814         }
815
816         /* restore the fsr and emulate exceptions as needed */
817         if ((fsr & FSR_CEXC) & (fsr >> 23)) {
818                 __fenv_setfsr32(&fsr);
819                 if (fsr & FSR_OFC) {
820                         dummy = huge;
821                         dummy *= huge;
822                 } else if (fsr & FSR_UFC) {
823                         dummy = tiny;
824                         if (fsr & FSR_NXC)
825                                 dummy *= tiny;
826                         else
827                                 dummy -= tiny2;
828                 } else {
829                         dummy = huge;
830                         dummy += tiny;
831                 }
832         } else {
833                 fsr |= (fsr & 0x1f) << 5;
834                 __fenv_setfsr32(&fsr);
835         }
836         return (zz.q);
837 }
______unchanged_portion_omitted_

1225 #else
1226 #error Unknown architecture
1227 #endif
```

****
2009 Sun May 4 03:06:35 2014
new/usr/src/lib/libm/common/m9x/fmax.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 fmax = __fmax
#endif

/*
 * fmax(x,y) returns the larger of x and y.  If just one of the
 * arguments is NaN, fmax returns the other argument.  If both
 * arguments are NaN, fmax returns NaN.
 *
 * See fmaxf.c for a discussion of implementation trade-offs.
 */

#include "libm.h"       /* for isgreaterequal macro */
#include <fenv.h>

double
__fmax(double x, double y) {
        union {
                unsigned i[2];
                double d;
        } xx, yy;
        unsigned s;

        /* if y is nan, replace it by x */
        if (y != y)
                y = x;

        /* if x is nan, replace it by y */
        if (x != x)
                x = y;

        /* At this point, x and y are either both numeric, or both NaN */
        if (!isnan(x) && !isgreaterequal(x, y))
```



```
                /* if x is less than y or x and y are unordered, replace x by y */
#if defined(COMPARISON_MACRO_BUG)
                if (x < y)
#else
                if (!isgreaterequal(x, y))
#endif
                        x = y;

                /*
                 * clear the sign of the result if either x or y has its sign clear
                 * now x and y are either both NaN or both numeric; clear the
                 * sign of the result if either x or y has its sign clear
                 */
                xx.d = x;
                yy.d = y;
#if defined(__sparc)
                s = ~(xx.i[0] & yy.i[0]) & 0x80000000;
                xx.i[0] &= ~s;
#elif defined(__x86)
                s = ~(xx.i[1] & yy.i[1]) & 0x80000000;
                xx.i[1] &= ~s;
#else
#error Unknown architecture
#endif

                return (xx.d);
}
```

*_____unchanged_portion_omitted_*

```
**********************************************************
    4325 Sun May  4 03:06:36 2014
new/usr/src/lib/libm/common/m9x/fmaxf.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 fmaxf = __fmaxf
#endif

/*
 * fmax(x,y) returns the larger of x and y.  If just one of the
 * arguments is NaN, fmax returns the other argument.  If both
 * arguments are NaN, fmax returns NaN (ideally, one of the
 * argument NaNs).
 *
 * C99 does not require that fmax(-0,+0) = fmax(+0,-0) = +0, but
 * ideally fmax should satisfy this.
 *
 * C99 makes no mention of exceptions for fmax.  I suppose ideally
 * either fmax never raises any exceptions or else it raises the
 * invalid operation exception if and only if some argument is a
 * signaling NaN.  In the former case, fmax should always return
 * one of its arguments.  In the latter, fmax shouldn't return a
 * signaling NaN, although when both arguments are signaling NaNs,
 * this ideal is at odds with the stipulation that fmax should
 * always return one of its arguments.
 *
 * Commutativity of fmax follows from the properties listed above
 * except when both arguments are NaN.  In that case, fmax may be
 * declared commutative by fiat because there is no portable way
 * to tell different NaNs apart.  Ideally fmax would be truly com-
 * mutative for all arguments.
 *
 * On SPARC V8, fmax must involve tests and branches.  Ideally,
 * an implementation on SPARC V9 should avoid branching, using
 * conditional moves instead where necessary, and be as efficient
 * as possible in its use of other resources.
 *
```



```
 * It appears to be impossible to attain all of the aforementioned
 * ideals simultaneously.  The implementation below satisfies the
 * following (on SPARC):
 *
 * 1. fmax(x,y) returns the larger of x and y if neither x nor y
 *    is NaN and the non-NaN argument if just one of x or y is NaN.
 *    If both x and y are NaN, fmax(x,y) returns x unchanged.
 * 2. fmax(-0,+0) = fmax(+0,-0) = +0.
 * 3. If either argument is a signaling NaN, fmax raises the invalid
 *    operation exception.  Otherwise, it raises no exceptions.
 */

#include "libm.h"       /* for isgreaterequal macro */

float
__fmaxf(float x, float y) {
        /*
         * On SPARC v8plus/v9, this could be implemented as follows
         * (assuming %f0 = x, %f1 = y, return value left in %f0):
         *
         * fcmps        %fcc0,%f1,%f1
         * fmovsu       %fcc0,%f0,%f1
         * fcmps        %fcc0,%f0,%f1
         * fmovsul      %fcc0,%f1,%f0
         * st           %f0,[x]
         * st           %f1,[y]
         * ld           [x],%l0
         * ld           [y],%l1
         * and          %l0,%l1,%l2
         * sethi        %hi(0x80000000),%l3
         * andn         %l3,%l2,%l2
         * andn         %l0,%l2,%l0
         * st           %l0,[x]
         * ld           [x],%f0
         *
         * If VIS instructions are available, use this code instead:
         *
         * fcmps        %fcc0,%f1,%f1
         * fmovsu       %fcc0,%f0,%f1
         * fcmps        %fcc0,%f0,%f1
         * fmovsul      %fcc0,%f1,%f0
         * fands        %f0,%f1,%f2
         * fzeros       %f3
         * fnegs        %f3,%f3
         * fandnot2s %f3,%f2,%f2
         * fandnot2s %f0,%f2,%f0
         *
         * If VIS 3.0 instructions are available, use this:
         *
         * flcmps       %fcc0,%f0,%f1
         * fmovslg      %fcc0,%f1,%f0   ! move if %fcc0 is 1 or 2
         */

        union {
                unsigned i;
                float f;
        } xx, yy;
        unsigned s;

        /* if y is nan, replace it by x */
        if (y != y)
                y = x;

        /* if x is nan, replace it by y */
        if (x != x)
                x = y;
```

```
130         /* At this point, x and y are either both numeric, or both NaN */
131         if (!isnan(x) && !isgreaterequal(x, y))
130         /* if x is less than y or x and y are unordered, replace x by y */
131 #if defined(COMPARISON_MACRO_BUG)
132         if (x < y)
133 #else
134         if (!isgreaterequal(x, y))
135 #endif
132                 x = y;

134         /*
135          * clear the sign of the result if either x or y has its sign clear
139          * now x and y are either both NaN or both numeric; clear the
140          * sign of the result if either x or y has its sign clear
136          */
137         xx.f = x;
138         yy.f = y;
139         s = ~(xx.i & yy.i) & 0x80000000;
140         xx.i &= ~s;

142         return (xx.f);
143 }
```

_____unchanged_portion_omitted_

```
**********************************************************
    1874 Sun May  4 03:06:38 2014
new/usr/src/lib/libm/common/m9x/fmaxl.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 fmaxl = __fmaxl
#endif

#include "libm.h"       /* for isgreaterequal macro */

long double
__fmaxl(long double x, long double y) {
        union {
#if defined(__sparc)
                unsigned i[4];
#elif defined(__x86)
                unsigned i[3];
#else
#error Unknown architecture
#endif
                long double ld;
        } xx, yy;
        unsigned s;

        /* if y is nan, replace it by x */
        if (y != y)
                y = x;

        /* if x is nan, replace it by y */
        if (x != x)
                x = y;

        /* At this point, x and y are either both numeric, or both NaN */
        if (!isnan(x) && !isgreaterequal(x, y))
        /* if x is less than y or x and y are unordered, replace x by y */
#if defined(COMPARISON_MACRO_BUG)
        if (x != x || x < y)
#else
        if (!isgreaterequal(x, y))
#endif
                x = y;

        /*
         * clear the sign of the result if either x or y has its sign clear
         * now x and y are either both NaN or both numeric; clear the
         * sign of the result if either x or y has its sign clear
         */
        xx.ld = x;
        yy.ld = y;
#if defined(__sparc)
        s = ~(xx.i[0] & yy.i[0]) & 0x80000000;
        xx.i[0] &= ~s;
#elif defined(__x86)
        s = ~(xx.i[2] & yy.i[2]) & 0x8000;
        xx.i[2] &= ~s;
#else
#error Unknown architecture
#endif

        return (xx.ld);
}
```



```
_____unchanged_portion_omitted_
```

```
**********************************************************
    2006 Sun May  4 03:06:39 2014
new/usr/src/lib/libm/common/m9x/fmin.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 fmin = __fmin
#endif

/*
 * fmin(x,y) returns the smaller of x and y.  If just one of the
 * arguments is NaN, fmin returns the other argument.  If both
 * arguments are NaN, fmin returns NaN.
 *
 * See fmaxf.c for a discussion of implementation trade-offs.
 */

#include "libm.h"       /* for islessequal macro */

#include "fenv_inlines.h"
#include <stdio.h>
#endif /* ! codereview */
#include <sys/isa_defs.h>

double
__fmin(double x, double y) {
        union {
                unsigned i[2];
                double d;
        } xx, yy;
        unsigned s;

        /* if y is nan, replace it by x */
        if (y != y)
                y = x;

        /* if x is nan, replace it by y */
        if (x != x)
```



```
                x = y;

        /* At this point, x and y are either both numeric, or both NaN */
        if (!isnan(x) && !islessequal(x, y))
        /* if x is greater than y or x and y are unordered, replace x by y */
#if defined(COMPARISON_MACRO_BUG)
        if (x != x || x > y)
#else
        if (!islessequal(x, y))
#endif
                x = y;

        /*
         * set the sign of the result if either x or y has its sign set
         * now x and y are either both NaN or both numeric; set the
         * sign of the result if either x or y has its sign set
         */
        xx.d = x;
        yy.d = y;
#if defined(_BIG_ENDIAN)
        s = (xx.i[0] | yy.i[0]) & 0x80000000;
        xx.i[0] |= s;
#else
        s = (xx.i[1] | yy.i[1]) & 0x80000000;
        xx.i[1] |= s;
#endif

        return (xx.d);
}
______unchanged_portion_omitted_
```

```
**********************************************************
    2413 Sun May  4 03:06:40 2014
new/usr/src/lib/libm/common/m9x/fminf.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 fminf = __fminf
#endif

#include "libm.h"       /* for islessequal macro */

float
__fminf(float x, float y) {
        /*
         * On SPARC v8plus/v9, this could be implemented as follows
         * (assuming %f0 = x, %f1 = y, return value left in %f0):
         *
         * fcmps        %fcc0,%f1,%f1
         * fmovsu       %fcc0,%f0,%f1
         * fcmps        %fcc0,%f0,%f1
         * fmovsug      %fcc0,%f1,%f0
         * st           %f0,[x]
         * st           %f1,[y]
         * ld           [x],%l0
         * ld           [y],%l1
         * or           %l0,%l1,%l2
         * sethi        %hi(0x80000000),%l3
         * and          %l3,%l2,%l2
         * or           %l0,%l2,%l0
         * st           %l0,[x]
         * ld           [x],%f0
         *
         * If VIS instructions are available, use this code instead:
         *
         * fcmps        %fcc0,%f1,%f1
         * fmovsu       %fcc0,%f0,%f1
         * fcmps        %fcc0,%f0,%f1
         * fmovsug      %fcc0,%f1,%f0
```



```
         * fors         %f0,%f1,%f2
         * fzeros       %f3
         * fnegs        %f3,%f3
         * fands        %f3,%f2,%f2
         * fors         %f0,%f2,%f0
         *
         * If VIS 3.0 instructions are available, use this:
         *
         * flcmps       %fcc0,%f0,%f1
         * fmovsge      %fcc0,%f1,%f0   ! move if %fcc0 is 0 or 2
         */

        union {
                unsigned i;
                float f;
        } xx, yy;
        unsigned s;

        /* if y is nan, replace it by x */
        if (y != y)
                y = x;

        /* if x is nan, replace it by y */
        if (x != x)
                x = y;

        /* At this point, x and y are either both numeric, or both NaN */
        if (!isnan(x) && !islessequal(x, y))
/* if x is greater than y or x and y are unordered, replace x by y */
#if defined(COMPARISON_MACRO_BUG)
        if (x > y)
#else
        if (!islessequal(x, y))
#endif
                x = y;

        /*
         * set the sign of the result if either x or y has its sign set
         * now x and y are either both NaN or both numeric; set the
         * sign of the result if either x or y has its sign set
         */
        xx.f = x;
        yy.f = y;
        s = (xx.i | yy.i) & 0x80000000;
        xx.i |= s;

        return (xx.f);
}
```

_____***unchanged_portion_omitted_***

```
**********************************************************
    1860 Sun May  4 03:06:43 2014
new/usr/src/lib/libm/common/m9x/fminl.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 fminl = __fminl
#endif

#include "libm.h"       /* for islessequal macro */

long double
__fminl(long double x, long double y) {
        union {
#if defined(__sparc)
                unsigned i[4];
#elif defined(__x86)
                unsigned i[3];
#else
#error Unknown architecture
#endif
                long double ld;
        } xx, yy;
        unsigned s;

        /* if y is nan, replace it by x */
        if (y != y)
                y = x;

        /* if x is nan, replace it by y */
        if (x != x)
                x = y;

        /* At this point, x and y are either both numeric, or both NaN */
        if (!isnan(x) && !islessequal(x, y))
        /* if x is greater than y or x and y are unordered, replace x by y */
#if defined(COMPARISON_MACRO_BUG)
        if (x != x || x > y)
#else
        if (!islessequal(x, y))
#endif
                x = y;

        /*
         * set the sign of the result if either x or y has its sign set
         * now x and y are either both NaN or both numeric; set the
         * sign of the result if either x or y has its sign set
         */
        xx.ld = x;
        yy.ld = y;
#if defined(__sparc)
        s = (xx.i[0] | yy.i[0]) & 0x80000000;
        xx.i[0] |= s;
#elif defined(__x86)
        s = (xx.i[2] | yy.i[2]) & 0x8000;
        xx.i[2] |= s;
#else
#error Unknown architecture
#endif

        return (xx.ld);
}
```



```
_____unchanged_portion_omitted_
```

```
**********************************************************
    4344 Sun May  4 03:06:44 2014
new/usr/src/lib/libm/common/m9x/llrintl.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 llrintl = __llrintl
#if defined(__sparcv9) || defined(__amd64)
#pragma weak lrintl = __llrintl
#pragma weak __lrintl = __llrintl
#endif
#endif

#include "libm.h"

#if defined(__sparc)

#include "fma.h"
#include "fenv_inlines.h"

long long
llrintl(long double x) {
        union {
                unsigned i[4];
                long double q;
        } xx;
        union {
                unsigned i[2];
                long long l;
        } zz;
        union {
                unsigned i;
                float f;
        } tt;
        unsigned int hx, sx, frac, fsr;
        unsigned int hx, sx, frac;
        unsigned int fsr;
        int rm, j;
```



```
        volatile float dummy;

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

        /* handle trivial cases */
        if (hx > 0x403e0000) { /* |x| > 2^63 + ... or x is nan */
                /* convert an out-of-range float */
                tt.i = sx | 0x7f000000;
                return ((long long) tt.f);
        } else if ((hx | xx.i[1] | xx.i[2] | xx.i[3]) == 0) /* x is zero */
                return (0LL);

        /* get the rounding mode */
        __fenv_getfsr32(&fsr);
        rm = fsr >> 30;

        /* flip the sense of directed roundings if x is negative */
        if (sx)
                rm ^= rm >> 1;

        /* handle |x| < 1 */
        if (hx < 0x3fff0000) {
                dummy = 1.0e30f; /* x is nonzero, so raise inexact */
                dummy += 1.0e-30f;
                if (rm == FSR_RP || (rm == FSR_RN && (hx >= 0x3ffe0000 &&
                        ((hx & 0xffff) | xx.i[1] | xx.i[2] | xx.i[3]))))
                        return (sx ? -1LL : 1LL);
                return (0LL);
        }

        /* extract the integer and fractional parts of x */
        j = 0x406f - (hx >> 16);
        xx.i[0] = 0x10000 | (xx.i[0] & 0xffff);
        if (j >= 96) {
                zz.i[0] = 0;
                zz.i[1] = xx.i[0] >> (j - 96);
                frac = ((xx.i[0] << 1) << (127 - j)) | (xx.i[1] >> (j - 96));
                if (((xx.i[1] << 1) << (127 - j)) | xx.i[2] | xx.i[3])
                        frac |= 1;
        } else if (j >= 64) {
                zz.i[0] = xx.i[0] >> (j - 64);
                zz.i[1] = ((xx.i[0] << 1) << (95 - j)) | (xx.i[1] >> (j - 64));
                frac = ((xx.i[1] << 1) << (95 - j)) | (xx.i[2] >> (j - 64));
                if (((xx.i[2] << 1) << (95 - j)) | xx.i[3])
                        frac |= 1;
        } else {
                zz.i[0] = ((xx.i[0] << 1) << (63 - j)) | (xx.i[1] >> (j - 32));
                zz.i[1] = ((xx.i[1] << 1) << (63 - j)) | (xx.i[2] >> (j - 32));
                frac = ((xx.i[2] << 1) << (63 - j)) | (xx.i[3] >> (j - 32));
                if ((xx.i[3] << 1) << (63 - j))
                        frac |= 1;
        }

        /* round */
        if (frac && (rm == FSR_RP || (rm == FSR_RN && (frac > 0x80000000u ||
                (frac == 0x80000000 && (zz.i[1] & 1)))))) {
                if (++zz.i[1] == 0)
                        zz.i[0]++;
        }

        /* check for result out of range (note that z is |x| at this point) */
        if (zz.i[0] > 0x80000000u || (zz.i[0] == 0x80000000 && (zz.i[1] ||
                !sx))) {
                tt.i = sx | 0x7f000000;
```

```
 127                 return ((long long) tt.f);
 128         }

 130         /* raise inexact if need be */
 131         if (frac) {
 132                 dummy = 1.0e30F;
 133                 dummy += 1.0e-30F;
 134         }

 136         /* negate result if need be */
 137         if (sx) {
 138                 zz.i[0] = ~zz.i[0];
 139                 zz.i[1] = -zz.i[1];
 140                 if (zz.i[1] == 0)
 141                         zz.i[0]++;
 142         }
 143         return (zz.l);
 144 }
```

unchanged_portion_omitted_

```
**********************************************************
    3910 Sun May  4 03:06:46 2014
new/usr/src/lib/libm/common/m9x/lrintl.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 lrintl = __lrintl
#endif

#include <sys/isa_defs.h>       /* _ILP32 */
#include "libm.h"

#if defined(_ILP32)
#if defined(__sparc)

#include "fma.h"
#include "fenv_inlines.h"

long
lrintl(long double x) {
        union {
                unsigned int i[4];
                unsigned i[4];
                long double q;
        } xx;
        union {
                unsigned int i;
                unsigned i;
                float f;
        } tt;
        unsigned int hx, sx, frac, l, fsr;
        unsigned hx, sx, frac, l;
        unsigned int fsr;
        int rm, j;
        volatile float dummy;

        xx.q = x;
        sx = xx.i[0] & 0x80000000;
```



```
        hx = xx.i[0] & ~0x80000000;

        /* handle trivial cases */
        if (hx > 0x401e0000) { /* |x| > 2^31 + ... or x is nan */
                /* convert an out-of-range float */
                tt.i = sx | 0x7f000000;
                return ((long) tt.f);
        } else if ((hx | xx.i[1] | xx.i[2] | xx.i[3]) == 0) /* x is zero */
                return (0L);

        /* get the rounding mode */
        __fenv_getfsr32(&fsr);
        rm = fsr >> 30;

        /* flip the sense of directed roundings if x is negative */
        if (sx)
                rm ^= rm >> 1;

        /* handle |x| < 1 */
        if (hx < 0x3fff0000) {
                dummy = 1.0e30F; /* x is nonzero, so raise inexact */
                dummy += 1.0e-30F;
                if (rm == FSR_RP || (rm == FSR_RN && (hx >= 0x3ffe0000 &&
                        ((hx & 0xffff) | xx.i[1] | xx.i[2] | xx.i[3]))))
                        return (sx ? -1L : 1L);
                return (0L);
        }

        /* extract the integer and fractional parts of x */
        j = 0x406f - (hx >> 16);                /* 91 <= j <= 112 */
        xx.i[0] = 0x10000 | (xx.i[0] & 0xffff);
        if (j >= 96) {                          /* 96 <= j <= 112 */
                l = xx.i[0] >> (j - 96);
                frac = ((xx.i[0] << 1) << (127 - j)) | (xx.i[1] >> (j - 96));
                if (((xx.i[1] << 1) << (127 - j)) | xx.i[2] | xx.i[3])
                        frac |= 1;
        } else {                                /* 91 <= j <= 95 */
                l = (xx.i[0] << (96 - j)) | (xx.i[1] >> (j - 64));
                frac = (xx.i[1] << (96 - j)) | (xx.i[2] >> (j - 64));
                if ((xx.i[2] << (96 - j)) | xx.i[3])
                        frac |= 1;
        }

        /* round */
        if (frac && (rm == FSR_RP || (rm == FSR_RN && (frac > 0x80000000U ||
                (frac == 0x80000000 && (l & 1))))))
                l++;

        /* check for result out of range (note that z is |x| at this point) */
        if (l > 0x80000000U || (l == 0x80000000U && !sx)) {
                tt.i = sx | 0x7f000000;
                return ((long) tt.f);
        }

        /* raise inexact if need be */
        if (frac) {
                dummy = 1.0e30F;
                dummy += 1.0e-30F;
        }

        /* negate result if need be */
        if (sx)
                l = -l;
        return ((long) l);
}
```

_____***unchanged_portion_omitted_***

```
**********************************************************
    1527 Sun May  4 03:06:48 2014
new/usr/src/lib/libm/common/m9x/nan.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 nan = __nan
#endif

/*
 *  nan(c) returns a NaN.  This implementation ignores c.
 */

#include "libm.h"
#include <sys/isa_defs.h>

#if defined(__sparc)
#if defined(_BIG_ENDIAN)

static const union {
        unsigned i[2];
        double d;
} __nan_union = { 0x7fffffff, 0xffffffff };

#elif defined(__i386) || defined(__amd64)
#else

static const union {
        unsigned i[2];
        double d;
} __nan_union = { 0xffffffff, 0x7fffffff };

#else
#error Unknown architecture
#endif /* ! codereview */
#endif

/* ARGSUSED0 */
```



```
double
__nan(const char *c) {
        return (__nan_union.d);
}
```

****
5109 Sun May 4 03:06:50 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  4 03:06:51 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  4 03:06:53 2014
new/usr/src/lib/libm/common/m9x/scalblnl.c
**********************************************************
______unchanged_portion_omitted_
```

****
68985 Sun May 4 03:06:55 2014
new/usr/src/lib/libm/common/m9x/tgamma.c
****
_____unchanged_portion_omitted_

```
1392 /* INDENT OFF */
1393 static const double
1394         /* 0.134861805732790769689793935774652917006 */
1395         t0z1   =  0.13486180573279077377 08,
1396         t0z1_l = -4.08100777085782990225 31e-18,
1397         /* 0.461632144968362341262659542325721328468 */
1398         t0z2   =  0.46163214496836235678 50,
1399         t0z2_l = -1.55223481628586768905 21e-17,
1400         /* 0.819773101100500601787868704921606996312 */
1401         t0z3   =  0.81977310110050061187 08,
1402         t0z3_l = -1.00829451224871034983 25e-17;
1403         /* 1.134861805732790769689793935774652917006 */
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 4 03:06:56 2014
new/usr/src/lib/libm/common/m9x/tgammaf.c
****
_____unchanged_portion_omitted_

```
 390 /* INDENT OFF */
 391 static const double
 392 t0z1 = 0.13486180573279076968979393577465291700 6,
 393 t0z2 = 0.46163214496836234126265954232572132846 8,
 394 t0z3 = 0.81977310110050060178786870492160699631 2;
 395         /* 1.13486180573279076968979393577465291700 6 */
 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  4 03:06:58 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   =  0.134861805732790769677938505499703580 8810L,
t0z1_l =  1.185543027494933612539271715025737961 4654e-20L,
#else
t0z1   =  0.134861805732790769689793935774652916 8654L,
t0z1_l =  1.410208858867687941873916448615951467 4310e-37L,
#endif
        /* 0.461632144968362341262659542325721328468196204006446351295 99 */
#if defined(__x86)
t0z2   =  0.461632144968362341253811584329547201 8326L,
t0z2_l =  8.847957996174126635585323050392617470306 40e-21L,
#else
t0z2   =  0.461632144968362341262659542325721321 3343318L,
t0z2_l =  5.035011623296163804653026664809162716111 01e-36L,
#endif
        /* 0.819773101100500601787868704921606996311744078462451791195 86 */
#if defined(__x86)
t0z3   =  0.819773101100500601787733623293519258368 17L,
t0z3_l =  1.350816280877379435658077052534574556256 230e-22L
#else
t0z3   =  0.819773101100500601787868704921606951695744 9L,
t0z3_l =  4.461599916947014419045492615933551648857380e-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_***

```
**********************************************************
    5896 Sun May  4 03:06:59 2014
new/usr/src/lib/libm/i386/src/libm_inlines.h
**********************************************************
```

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

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

/*
 * Copyright 2011, Richard Lowe
 */

/* Functions in this file are duplicated in locallibm.il.  Keep them in sync */
/* Functions in this file are duplicated in libm.m4.  Keep them in sync */

#ifndef _LIBM_INLINES_H
#define _LIBM_INLINES_H

#ifdef __GNUC__

#ifdef __cplusplus
extern "C" {
#endif

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

#define _LO_WORD(x)     ((uint32_t *)&x)[0]
#define _HI_WORD(x)     ((uint32_t *)&x)[1]
#define _HIER_WORD(x)   ((uint32_t *)&x)[2]

extern __inline__ double
__ieee754_sqrt(double a)
{
        double ret;

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

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



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

extern __inline__ double
__ieee754_sqrt(double a)
__d_sqrt_(double *a)
{
        return (__inline_sqrt(a));
        double ret;

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

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

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

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

        __asm__ __volatile__(
            "andl $0x7fffffff,%1\n\t"
            "cmpl $0x43300000,%1\n\t"
                "andl $0x7fffffff,%2\n\t"
                "cmpl $0x43300000,%2\n\t"
            "jae  1f\n\t"
            "frndint\n\t"
            "1: fwait\n\t"
            : "+t" (a), "+&r" (_HI_WORD(a))
            :
            : "cc");
                : "=t" (ret)
                : "0" (a), "r" (_HI_WORD(a)));

        return (a);
        return (ret);
}

extern __inline__ short
__inline_fstsw(void)
{
        short ret;

        __asm__ __volatile__("fstsw %0\n\t" : "=r" (ret));
        return (ret);
}
______unchanged_portion_omitted_

extern __inline__ double
ceil(double d)
{
        double ret;
        short rd = __swap87RD(fp_positive);
```

```
138         __asm__ __volatile__("frndint" : "+t" (d), "+r" (rd) : : "cc");
161         __asm__ __volatile__("frndint" : "=t" (ret) : "0" (d));
139         __swap87RD(rd);

141         return (d);
164         return (ret);
142 }

144 extern __inline__ double
145 copysign(double d1, double d2)
146 {
147         __asm__ __volatile__(
148             "andl $0x7fffffff,%0\n\t"   /* %0 <-- hi_32(abs(d)) */
149             "andl $0x80000000,%1\n\t"   /* %1[31] <-- sign_bit(d2) */
150             "orl  %1,%0\n\t"            /* %0 <-- hi_32(copysign(x,y)) */
151             : "+&r" (_HI_WORD(d1)), "+r" (_HI_WORD(d2))
152             :
153             : "cc");
174             : "+r" (_HI_WORD(d1))
175             : "r" (_HI_WORD(d2)));

155         return (d1);
156 }

158 extern __inline__ double
159 fabs(double d)
160 {
161         __asm__ __volatile__("fabs\n\t" : "+t" (d) : : "cc");
162         return (d);
183         double ret;

185         __asm__ __volatile__("fabs\n\t" : "=t" (ret) : "0" (d));
186         return (ret);
163 }

165 extern __inline__ float
166 fabsf(float d)
167 {
168         __asm__ __volatile__("fabs\n\t" : "+t" (d) : : "cc");
169         return (d);
192         float ret;

194         __asm__ __volatile__("fabs\n\t" : "=t" (ret) : "0" (d));
195         return (ret);
170 }

172 extern __inline__ long double
173 fabsl(long double d)
174 {
175         __asm__ __volatile__("fabs\n\t" : "+t" (d) : : "cc");
176         return (d);
201         long double ret;

203         __asm__ __volatile__("fabs\n\t" : "=t" (ret) : "0" (d));
204         return (ret);
177 }

179 extern __inline__ int
180 finite(double d)
181 {
182         int ret = _HI_WORD(d);
210         int ret;

184         __asm__ __volatile__(
185             "notl %0\n\t"
186             "andl $0x7ff00000,%0\n\t"
```



```
187             "negl %0\n\t"
188             "shrl $31,%0\n\t"
189             : "+r" (ret)
190             :
191             : "cc");
213             "notl %1\n\t"
214             "andl $0x7ff00000,%1\n\t"
215             "negl %1\n\t"
216             "shrl $31,%1\n\t"
217             : "=r" (ret)
218             : "0" (_HI_WORD(d)));
192         return (ret);
193 }

195 extern __inline__ double
196 floor(double d)
197 {
225         double ret;
198         short rd = __swap87RD(fp_negative);

200         __asm__ __volatile__("frndint" : "+t" (d), "+r" (rd) : : "cc");
228         __asm__ __volatile__("frndint" : "=t" (ret) : "0" (d));
201         __swap87RD(rd);

203         return (d);
231         return (ret);
232 }

234 /*
235  *      branchless __isnan
236  *      ((0x7ff00000-[((lx|-lx)>>31)&1]|ahx)>>31)&1 = 1 iff x is NaN
237  */
238 extern __inline__ int
239 isnan(double d)
240 {
241         int ret;

243         __asm__ __volatile__(
244         "movl %1,%%ecx\n\t"
245         "negl %%ecx\n\t"          /* ecx <-- -lo_32(x) */
246         "orl  %%ecx,%1\n\t"
247         "shrl $31,%1\n\t"         /* 1 iff lx != 0 */
248         "andl $0x7fffffff,%2\n\t" /* ecx <-- hi_32(abs(x)) */
249         "orl  %2,%1\n\t"
250         "subl $0x7ff00000,%1\n\t"
251         "negl %1\n\t"
252         "shrl $31,%1\n\t"
253         : "=r" (ret)
254         : "0" (_HI_WORD(d)), "r" (_LO_WORD(d))
255         : "ecx");

257         return (ret);
204 }

206 extern __inline__ int
207 isnanf(float f)
208 {
263         int ret;

209         __asm__ __volatile__(
210             "andl $0x7fffffff,%0\n\t"
211             "negl %0\n\t"
212             "addl $0x7f800000,%0\n\t"
213             "shrl $31,%0\n\t"
214             : "+r" (f)
215             :
```

```
216             : "cc");
270             : "=r" (ret)
271             : "0" (f));

273         return (ret);
274 }

276 extern __inline__ int
277 isinf(double d)
278 {
279         int ret;

218         return (f);
281         __asm__ __volatile__(
282         "andl $0x7fffffff,%1\n\t"           /* set first bit to 0 */
283         "cmpl $0x7ff00000,%1\n\t"
284         "pushfl\n\t"
285         "popl %0\n\t"
286         "cmpl $0,%2\n\t"                /* is lo_32(x) = 0? */
287         "pushfl\n\t"
288         "popl %2\n\t"   /* bit 6 of ecx <-- lo_32(x) == 0 */
289         "andl %2,%0\n\t"
290         "andl $0x40,%0\n\t"
291         "shrl $6,%0\n\t"
292         : "=r" (ret)
293         : "0" (_HI_WORD(d)), "r" (_LO_WORD(d)));

295         return (ret);
219 }

221 extern __inline__ double
222 rint(double a) {
223     return (__inline_rint(a));
300         double ret;

302         __asm__ __volatile__(
303                 "andl $0x7fffffff,%2\n\t"
304                 "cmpl $0x43300000,%2\n\t"
305                 "jae  1f\n\t"
306                 "frndint\n\t"
307                 "1: fwait\n\t"
308                 : "=t" (ret)
309                 : "0" (a), "r" (_HI_WORD(a)));

311         return (ret);
224 }

226 extern __inline__ double
227 scalbn(double d, int n)
228 {
229         double dummy;
317         double ret, dummy;

231         __asm__ __volatile__(
232             "fildl %2\n\t"      /* Convert N to extended */
320             "fildl %3\n\t"      /* Convert N to extended */
233             "fxch\n\t"
234             "fscale\n\t"
235             : "+t" (d), "=u" (dummy)
236             : "m" (n)
237             : "cc");
323             : "=t" (ret), "=u" (dummy)
324             : "0" (d), "m" (n));

239         return (d);
326         return (ret);
```



```
240 }
______unchanged_portion_omitted_

254 extern __inline__ double
255 sqrt(double d)
256 {
257         return (__inline_sqrt(d));
344         double ret;
345         __asm__ __volatile__("fsqrt" : "=t" (ret) : "0" (d));
346         return (ret);
258 }

260 extern __inline__ float
261 sqrtf(float f)
262 {
263         return (__inline_sqrtf(f));
352         float ret;
353         __asm__ __volatile__("fsqrt" : "=t" (ret) : "0" (f));
354         return (ret);
264 }

266 extern __inline__ long double
267 sqrtl(long double ld)
268 {
269         __asm__ __volatile__("fsqrt" : "+t" (ld) : : "cc");
270         return (ld);
360         long double ret;
361         __asm__ __volatile__("fsqrt" : "=t" (ret) : "0" (ld));
362         return (ret);
271 }

273 extern __inline__ int
274 isnanl(long double ld)
275 {
276         int ret = _HIER_WORD(ld);
368         int ret;

278         __asm__ __volatile__(
279             "andl  $0x00007fff,%0\n\t"
280             "jz    1f\n\t"              /* jump if exp is all 0 */
281             "xorl  $0x00007fff,%0\n\t"
282             "jz    2f\n\t"              /* jump if exp is all 1 */
283             "testl $0x80000000,%1\n\t"
371             "andl  $0x00007fff,%1\n\t"
372             "jz    1f\n\t"            /* jump if __exp is all 0 */
373             "xorl  $0x00007fff,%1\n\t"
374             "jz    2f\n\t"            /* jump if __exp is all 1 */
375             "testl $0x80000000,%2\n\t"
284             "jz    3f\n\t"              /* jump if leading bit is 0 */
285             "movl  $0,%0\n\t"
377             "movl  $0,%1\n\t"
286             "jmp   1f\n\t"
287             "2:\n\t"                    /* note that %0 = 0 from before */
288             "cmpl  $0x80000000,%1\n\t"  /* what is first half of significand? */
379             "2:\n\t"                   /* note that %eax = 0 from before */
380             "cmpl  $0x80000000,%2\n\t" /* what is first half of __significand? *
289             "jnz   3f\n\t"              /* jump if not equal to 0x80000000 */
290             "testl $0xffffffff,%2\n\t"  /* is second half of significand 0? */
382             "testl $0xffffffff,%3\n\t" /* is second half of __significand 0? */
291             "jnz   3f\n\t"              /* jump if not equal to 0 */
292             "jmp   1f\n\t"
293             "3:\n\t"
294             "movl  $1,%0\n\t"
386             "movl  $1,%1\n\t"
295             "1:\n\t"
296             : "+&r" (ret)
```

```
297             : "r" (_HI_WORD(ld)), "r" (_LO_WORD(ld))
298             : "cc");
388             : "=r" (ret)
389             : "0" (_HIER_WORD(ld)), "r" (_HI_WORD(ld)), "r" (_LO_WORD(ld)));

300         return (ret);
301 }
```

*unchanged_portion_omitted_*

```
**********************************************************
    7507 Sun May  4 03:07:01 2014
new/usr/src/lib/libm/i386/src/locallibm.il
**********************************************************
/
/ CDDL HEADER START
/
/ The contents of this file are subject to the terms of the
/ Common Development and Distribution License (the "License").
/ You may not use this file except in compliance with the License.
/
/ You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
/ or http://www.opensolaris.org/os/licensing.
/ See the License for the specific language governing permissions
/ and limitations under the License.
/
/ When distributing Covered Code,  this CDDL HEADER in each
/ file and  the License file at usr/src/OPENSOLARIS.LICENSE.
/ If applicable, add the following below this CDDL HEADER, with the
/ fields enclosed by brackets "[]" 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.
/

/ Portions of this file are duplicated as GCC inline assembly in
/ libm_inlines.h.  Keep them in sync.

        .inline __ieee754_sqrt,0
        fldl    (%esp)
        fsqrt
        .end

        .inline __inline_rint,0
        fldl    (%esp)
        movl    4(%esp),%eax
        andl    $0x7fffffff,%eax
        cmpl    $0x43300000,%eax
        jae     1f
        frndint
1:
        fwait           / in case we jumped around the frndint
        .end

        .inline __inline_sqrtf,0
        flds    (%esp)
        fsqrt
        .end

        .inline __inline_sqrt,0
        fldl    (%esp)
        fsqrt
        .end

        .inline __inline_fstsw,0
        fstsw   %ax
        .end

/
/ 00 - 24 bits
/ 01 - reserved
```



```
/ 10 - 53 bits
/ 11 - 64 bits
/
        .inline __swapRP,0
        subl    $4,%esp
        fstcw   (%esp)
        movw    (%esp),%ax
        movw    %ax,%cx
        andw    $0xfcff,%cx
        movl    4(%esp),%edx            ///
        andl    $0x3,%edx
        shlw    $8,%dx
        orw     %dx,%cx
        movl    %ecx,(%esp)
        fldcw   (%esp)
        shrw    $8,%ax
        andl    $0x3,%eax
        addl    $4,%esp
        .end

/
/ 00 - Round to nearest, with even preferred
/ 01 - Round down
/ 10 - Round up
/ 11 - Chop
/
        .inline __swap87RD,0
        subl    $4,%esp
        fstcw   (%esp)
        movw    (%esp),%ax
        movw    %ax,%cx
        andw    $0xf3ff,%cx
        movl    4(%esp),%edx
        andl    $0x3,%edx
        shlw    $10,%dx
        orw     %dx,%cx
        movl    %ecx,(%esp)
        fldcw   (%esp)
        shrw    $10,%ax
        andl    $0x3,%eax
        addl    $4,%esp
        .end

/
/       Convert Top-of-Stack to long
/
        .inline __xtol,0
        subl    $8,%esp                 / 8 bytes of stack space
        fstcw   2(%esp)                 / byte[2:3] = old_cw
        movw    2(%esp),%ax
        andw    $0xf3ff,%ax
        orw     $0x0c00,%ax             / RD set to Chop
        movw    %ax,(%esp)              / byte[0:1] = new_cw
        fldcw   (%esp)                  / set new_cw
        fistpl  4(%esp)                 / byte[4:7] = converted long
        fstcw   (%esp)                  / restore old RD
        movw    (%esp),%ax
        andw    $0xf3ff,%ax
        movw    2(%esp),%dx
        andw    $0x0c00,%dx
        orw     %ax,%dx
        movw    %dx,2(%esp)
        fldcw   2(%esp)
        movl    4(%esp),%eax
        addl    $8,%esp
        .end
```

```
130         .inline __ceil,0
131         subl    $8,%esp
132         fstcw   (%esp)
133         fldl    8(%esp)                 ///
134         movw    (%esp),%cx
135         orw     $0x0c00,%cx
136         xorw    $0x0400,%cx
137         movw    %cx,4(%esp)
138         fldcw   4(%esp)                 / set RD = up
139         frndint
140         fstcw   4(%esp)                 / restore RD
141         movw    4(%esp),%dx
142         andw    $0xf3ff,%dx
143         movw    (%esp),%cx
144         andw    $0x0c00,%cx
145         orw     %dx,%cx
146         movw    %cx,(%esp)
147         fldcw   (%esp)
148         addl    $8,%esp
149         .end

151         .inline __copysign,0
152         movl    4(%esp),%eax            /// eax <-- hi_32(x)
153         movl    12(%esp),%ecx           /// ecx <-- hi_32(y)
154         andl    $0x7fffffff,%eax        / eax <-- hi_32(abs(x))
155         andl    $0x80000000,%ecx        / ecx[31] <-- sign_bit(y)
156         orl     %ecx,%eax               / eax <-- hi_32(__copysign(x,y))
157         movl    (%esp),%ecx             /// ecx <-- lo_32(x)
158                                         /       = lo_32(__copysign(x,y))
159         subl    $8,%esp                 / set up loading dock for result
160         movl    %ecx,(%esp)             / copy lo_32(result) to loading dock
161         movl    %eax,4(%esp)            / copy hi_32(result) to loading dock
162         fldl    (%esp)                  / load __copysign(x,y)
163         fwait                           / in case fldl causes exception
164         addl    $8,%esp                 / restore stack-pointer
165         .end

167         .inline __d_sqrt_,0
168         movl    (%esp),%eax
169         fldl    (%eax)
170         fsqrt
171         .end

173         .inline __fabs,0
174         fldl    (%esp)                  ///
175         fabs
176         .end

178         .inline __fabsf,0
179         flds    (%esp)
180         fabs
181         .end

183         .inline __fabsl,0
184         fldt    (%esp)
185         fabs
186         .end

188 /
189 /       branchless _finite
190 /
191         .inline _finite,0
192         movl    4(%esp),%eax            /// eax <-- hi_32(x)
193         notl    %eax                    / not(bexp) = 0 iff bexp = all 1's
194         andl    $0x7ff00000,%eax
```



```
195         negl    %eax
196         shrl    $31,%eax
197         .end

199         .inline __floor,0
200         subl    $8,%esp
201         fstcw   (%esp)
202         fldl    8(%esp)                 ///
203         movw    (%esp),%cx
204         orw     $0x0c00,%cx
205         xorw    $0x0800,%cx
206         movw    %cx,4(%esp)
207         fldcw   4(%esp)                 / set RD = down
208         frndint
209         fstcw   4(%esp)                 / restore RD
210         movw    4(%esp),%dx
211         andw    $0xf3ff,%dx
212         movw    (%esp),%cx
213         andw    $0x0c00,%cx
214         orw     %dx,%cx
215         movw    %cx,(%esp)
216         fldcw   (%esp)                  / restore RD
217         addl    $8,%esp
218         .end

220 /
221 /       branchless __isnan
222 /       ((0x7ff00000-[((lx|-lx)>>31)&1]|ahx)>>31)&1 = 1 iff x is NaN
223 /
224         .inline __isnan,0
225         movl    (%esp),%eax             /// eax <-- lo_32(x)
226         movl    %eax,%ecx
227         negl    %ecx                    / ecx <-- -lo_32(x)
228         orl     %ecx,%eax
229         shrl    $31,%eax                / 1 iff lx != 0
230         movl    4(%esp),%ecx            /// ecx <-- hi_32(x)
231         andl    $0x7fffffff,%ecx        / ecx <-- hi_32(abs(x))
232         orl     %ecx,%eax
233         subl    $0x7ff00000,%eax
234         negl    %eax
235         shrl    $31,%eax
236         .end

220         .inline __isnanf,0
221         movl    (%esp),%eax
222         andl    $0x7fffffff,%eax
223         negl    %eax
224         addl    $0x7f800000,%eax
225         shrl    $31,%eax
226         .end

246         .inline __isinf,0
247         movl    4(%esp),%eax            / eax <-- hi_32(x)
248         andl    $0x7fffffff,%eax        / set first bit to 0
249         cmpl    $0x7ff00000,%eax
250         pushfl
251         popl    %eax
252         cmpl    $0,(%esp)               / is lo_32(x) = 0?
253         pushfl
254         popl    %ecx                    / bit 6 of ecx <-- lo_32(x) == 0
255         andl    %ecx,%eax
256         andl    $0x40,%eax
257         shrl    $6,%eax
258         .end

229         .inline __isnormal,0
```

```
230                                     / TRUE iff (x is _finite, but
231                                     /           neither subnormal nor +/-0)
232                                     /      iff (0 < bexp(x) < 0x7ff)
233         movl    4(%esp),%eax        / eax <-- hi_32(x)
234         andl    $0x7ff00000,%eax    / eax[20..30]  <-- bexp(x),
235                                     / rest_of(eax) <-- 0
236         pushfl
237         popl    %ecx                / bit 6 of ecx <-- not bexp(x)
238         subl    $0x7ff00000,%eax
239         pushfl
240         popl    %eax                / bit 6 of eax <-- not bexp(x)
241         orl     %ecx,%eax
242         andl    $0x40,%eax
243         xorl    $0x40,%eax
244         shrl    $6,%eax
245         .end

247         .inline __issubnormal,0
248                                     / TRUE iff (bexp(x) = 0 and
249                                     /           frac(x) /= 0)
250         movl    $0,%eax
251         movl    4(%esp),%ecx        / ecx <-- hi_32(x)
252         andl    $0x7fffffff,%ecx    / ecx <-- hi_32(abs(x))
253         cmpl    $0x00100000,%ecx    / is bexp(x) = 0?
254         adcl    $0,%eax             / jump if bexp(x) = 0
255         orl     (%esp),%ecx         / = 0 iff sgnfcnd(x) = 0
256                                     /     iff x = +/- 0.0 here
257         pushfl
258         popl    %ecx
259         andl    $0x40,%ecx
260         xorl    $0x40,%ecx
261         shrl    $6,%ecx
262         andl    %ecx,%eax
263         .end

265         .inline __iszero,0
266         movl    4(%esp),%eax        / eax <-- hi_32(x)
267         andl    $0x7fffffff,%eax    / eax <-- hi_32(abs(x))
268         orl     (%esp),%eax         / = 0 iff x = +/- 0.0
269         pushfl
270         popl    %eax
271         andl    $0x40,%eax
272         shrl    $6,%eax
273         .end

275         .inline __r_sqrt_,0
276         movl    (%esp),%eax
277         flds    (%eax)
278         fsqrt
279         .end

281         .inline __rint,0
282         fldl    (%esp)
283         movl    4(%esp),%eax
284         andl    $0x7fffffff,%eax
285         cmpl    $0x43300000,%eax
286         jae     1f
287         frndint
288 1:
289         fwait                   / in case we jumped around frndint
290         .end

292         .inline __scalbn,0
293         fildl   8(%esp)             /// convert N to extended
294         fldl    (%esp)              /// push x
295         fscale
```



```
296         fstp    %st(1)
297         .end

299         .inline __signbit,0
300         movl    4(%esp),%eax            /// high part of x
301         shrl    $31,%eax
302         .end

304         .inline __signbitf,0
305         movl    (%esp),%eax
306         shrl    $31,%eax
307         .end

309         .inline __sqrt,0
310         fldl    (%esp)
311         fsqrt
312         .end

314         .inline __sqrtf,0
315         flds    (%esp)
316         fsqrt
317         .end

319         .inline __sqrtl,0
320         fldt    (%esp)
321         fsqrt
322         .end

324         .inline __isnanl,0
325         movl    8(%esp),%eax            / ax <-- sign bit and __exp
326         andl    $0x00007fff,%eax
327         jz      1f                      / jump if __exp is all 0
328         xorl    $0x00007fff,%eax
329         jz      2f                      / jump if __exp is all 1
330         testl   $0x80000000,4(%esp)
331         jz      3f                      / jump if leading bit is 0
332         movl    $0,%eax
333         jmp     1f
334 2:                                      / note that %eax = 0 from before
335         cmpl    $0x80000000,4(%esp)     / what is first half of __significand?
336         jnz     3f                      / jump if not equal to 0x80000000
337         testl   $0xffffffff,(%esp)      / is second half of __significand 0?
338         jnz     3f                      / jump if not equal to 0
339         jmp     1f
340 3:
341         movl    $1,%eax
342 1:
343         .end

345         .inline __f95_signf,0
346         sub     $4,%esp
347         mov     4(%esp),%edx
348         mov     (%edx),%eax
349         and     $0x7fffffff,%eax
350         mov     8(%esp),%edx
351         mov     (%edx),%ecx
352         and     $0x80000000,%ecx
353         or      %ecx,%eax
354         mov     %eax,(%esp)
355         flds    (%esp)
356         add     $4,%esp
357         .end

359         .inline __f95_sign,0
360         mov     (%esp),%edx
361         fldl    (%edx)
```

```
 362         fabs
 363         mov     4(%esp),%edx
 364         mov     4(%edx),%eax
 365         test    %eax,%eax
 366         jns     1f
 367         fchs
 368 1:
 369         .end
```

```
**********************************************************
    6972 Sun May  4 03:07:04 2014
new/usr/src/lib/libm/sparc/src/libm_inlines.h
**********************************************************
```

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

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

/*
 * Copyright 2011, Richard Lowe.
 */

/* Functions in this file are duplicated in locallibm.il.  Keep them in sync */
/* Functions in this file are duplicated in libm.m4.  Keep them in sync */

#ifndef _LIBM_INLINES_H
#define _LIBM_INLINES_H

#ifdef __GNUC__

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

#ifdef __cplusplus
extern "C" {
#endif

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

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

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

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



```
        return (ret);
}

extern __inline__ enum fp_class_type
fp_classf(float f)
{
        enum fp_class_type ret;
        uint32_t tmp;
#endif /* ! codereview */

        /* XXX: Separate input and output */
#endif /* ! codereview */
        __asm__ __volatile__(
            "sethi  %%hi(0x80000000),%1\n\t"
            "andncc %3,%1,%0\n\t"
            "sethi  %%hi(0x80000000),%%o2\n\t"
            "andncc %0,%%o2,%0\n\t"
            "bne    1f\n\t"
            "nop\n\t"
            "mov    0,%0\n\t"
            "ba 2f\n\t"         /* x is 0 */
            "nop\n\t"
            "1:\n\t"
            "sethi  %%hi(0x7f800000),%1\n\t"
            "andcc  %0,%1,%%g0\n\t"
            "sethi  %%hi(0x7f800000),%%o2\n\t"
            "andcc  %0,%%o2,%%g0\n\t"
            "bne    1f\n\t"
            "nop\n\t"
            "mov    1,%0\n\t"
            "ba     2f\n\t"     /* x is subnormal */
            "nop\n\t"
            "1:\n\t"
            "cmp    %0,%1\n\t"
            "cmp    %0,%%o2\n\t"
            "bge    1f\n\t"
            "nop\n\t"
            "mov    2,%0\n\t"
            "ba     2f\n\t"     /* x is normal */
            "nop\n\t"
            "1:\n\t"
            "bg     1f\n\t"
            "nop\n\t"
            "mov    3,%0\n\t"
            "ba     2f\n\t"     /* x is __infinity */
            "nop\n\t"
            "1:\n\t"
            "sethi  %%hi(0x00400000),%1\n\t"
            "andcc  %0,%1,%%g0\n\t"
            "sethi  %%hi(0x00400000),%%o2\n\t"
            "andcc  %0,%%o2,%%g0\n\t"
            "mov    4,%0\n\t"   /* x is quiet NaN */
            "bne    2f\n\t"
            "nop\n\t"
            "mov    5,%0\n\t"   /* x is signaling NaN */
            "2:\n\t"
            : "+r" (ret), "=&r" (tmp)
            : "r" (f)
            : "cc");
            : "=r" (ret)
            : "0" (f)
            : "o2");
        return (ret);
}

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

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

extern __inline__ enum fp_class_type
fp_class(double d)
{
        enum fp_class_type ret;
        uint32_t tmp;
#endif /* ! codereview */

        __asm__ __volatile__(
            "sethi %%hi(0x80000000),%1\n\t"     /* %1 gets 80000000 */
            "andn  %2,%1,%0\n\t"                /* %2-%0 gets abs(x) */
            "orcc  %0,%3,%%g0\n\t"              /* set cc as x is zero/nonzero *
            "sethi %%hi(0x80000000),%%o2\n\t" /* o2 gets 80000000 */
            "andn  %0,%%o2,%0\n\t"            /* o0-o1 gets abs(x) */
            "orcc  %0,%2,%%g0\n\t"            /* set cc as x is zero/nonzero */
            "bne   1f\n\t"                      /* branch if x is nonzero */
            "nop\n\t"
            "mov   0,%0\n\t"
            "ba    2f\n\t"                      /* x is 0 */
            "nop\n\t"
            "1:\n\t"
            "sethi %%hi(0x7ff00000),%1\n\t"     /* %1 gets 7ff00000 */
            "andcc %0,%1,%%g0\n\t"              /* cc set by __exp field of x */
            "sethi %%hi(0x7ff00000),%%o2\n\t" /* o2 gets 7ff00000 */
            "andcc %0,%%o2,%%g0\n\t"          /* cc set by __exp field of x */
            "bne   1f\n\t"                      /* branch if normal or max __exp
            "nop\n\t"
            "mov   1,%0\n\t"
            "ba    2f\n\t"                      /* x is subnormal */
            "nop\n\t"
            "1:\n\t"
            "cmp   %0,%1\n\t"
            "cmp   %0,%%o2\n\t"
            "bge   1f\n\t"                      /* branch if x is max __exp */
            "nop\n\t"
            "mov   2,%0\n\t"
            "ba    2f\n\t"                      /* x is normal */
            "nop\n\t"
            "1:\n\t"
            "andn  %0,%1,%0\n\t"                /* o0 gets msw __significand fie
            "orcc  %0,%3,%%g0\n\t"              /* set cc by OR __significand */
            "andn  %0,%%o2,%0\n\t"            /* o0 gets msw __significand field
            "orcc  %0,%2,%%g0\n\t"            /* set cc by OR __significand */
            "bne   1f\n\t"                      /* Branch if __nan */
            "nop\n\t"
            "mov   3,%0\n\t"
            "ba    2f\n\t"                      /* x is __infinity */
            "nop\n\t"
            "1:\n\t"
            "sethi %%hi(0x00080000),%1\n\t"
            "andcc %0,%1,%%g0\n\t"              /* set cc by quiet/sig bit */
            "sethi %%hi(0x00080000),%%o2\n\t"
            "andcc %0,%%o2,%%g0\n\t"          /* set cc by quiet/sig bit */
            "be    1f\n\t"                      /* Branch if signaling */
            "nop\n\t"
            "mov   4,%0\n\t"                    /* x is quiet NaN */
            "ba    2f\n\t"
            "nop\n\t"
            "1:\n\t"
            "mov   5,%0\n\t"                    /* x is signaling NaN */
            "2:\n\t"
            : "=&r" (ret), "=&r" (tmp)
            : "r" (_HI_WORD(d)), "r" (_LO_WORD(d))
            : "cc");
            : "=r" (ret)
```



```
            : "0" (_HI_WORD(d)), "r" (_LO_WORD(d))
            : "o2");

        return (ret);
}

extern __inline__ int
__swapEX(int i)
{
        int ret;
        uint32_t fsr;
        uint32_t tmp1, tmp2;
#endif /* ! codereview */

        __asm__ __volatile__(
            "and  %4,0x1f,%3\n\t"
            "sll  %3,5,%3\n\t"  /* shift input to aexc bit location */
            "and  %0,0x1f,%%o1\n\t"
            "sll  %%o1,5,%%o1\n\t"    /*  input to aexc bit location */
            ".volatile\n\t"
            "st   %%fsr,%1\n\t"
            "ld   %1,%0\n\t"    /* %0 = fsr */
            "andn %0,0x3e0,%4\n\t"
            "or   %3,%4,%3\n\t" /* %3 = new fsr */
            "st   %3,%1\n\t"
            "ld   %1,%%fsr\n\t"
            "st   %%fsr,%2\n\t"
            "ld   %2,%0\n\t"          /* = fsr */
            "andn %0,0x3e0,%%o2\n\t"
            "or   %%o1,%%o2,%%o1\n\t" /* o1 = new fsr */
            "st   %%o1,%2\n\t"
            "ld   %2,%%fsr\n\t"
            "srl  %0,5,%0\n\t"
            "and  %0,0x1f,%0\n\t"
            ".nonvolatile\n\t"
            : "=r" (ret), "=m" (fsr), "=r" (tmp1), "=r" (tmp2)
            : "r" (i)
            : "cc");
            : "=r" (ret)
            : "0" (i), "m" (fsr)
            : "o1", "o2");

        return (ret);
}
______unchanged_portion_omitted_

extern __inline__ enum fp_direction_type
__swapRD(enum fp_direction_type d)
{
        enum fp_direction_type ret;
        uint32_t fsr;
        uint32_t tmp1, tmp2, tmp3;
#endif /* ! codereview */

        __asm__ __volatile__(
            "and  %5,0x3,%0\n\t"
            "sll  %0,30,%2\n\t"         /* shift input to RD bit location */
            "and  %0,0x3,%0\n\t"
            "sll  %0,30,%%o1\n\t"      /* input to RD bit location */
            ".volatile\n\t"
            "st   %%fsr,%1\n\t"
            "ld   %1,%0\n\t"            /* %0 = fsr */
            "set  0xc0000000,%4\n\t"    /* mask of rounding direction bits */
            "andn %0,%4,%3\n\t"
            "or   %2,%3,%2\n\t"         /* %2 = new fsr */
            "st   %2,%1\n\t"
```

```
232             "ld   %1,%%fsr\n\t"
208             "st   %%fsr,%2\n\t"
209             "ld   %2,%0\n\t"           /* o0 = fsr */
210             "set  0xc0000000,%%o4\n\t" /* mask of rounding direction bits */
211             "andn %0,%%o4,%%o2\n\t"
212             "or   %%o1,%%o2,%%o1\n\t"  /* o1 = new fsr */
213             "st   %%o1,%2\n\t"
214             "ld   %2,%%fsr\n\t"
233             "srl  %0,30,%0\n\t"
234             "and  %0,0x3,%0\n\t"
235             ".nonvolatile\n\t"
236             : "=r" (ret), "=m" (fsr), "=r" (tmp1), "=r" (tmp2), "=r" (tmp3)
237             : "r" (d)
238             : "cc");
218             : "=r" (ret)
219             : "0" (d), "m" (fsr)
220             : "o1", "o2", "o4");

240         return (ret);
241 }

243 extern __inline__ int
244 __swapTE(int i)
245 {
246         int ret;
247         uint32_t fsr, tmp1, tmp2;
229         uint32_t fsr;

249         __asm__ __volatile__(
250             "and  %4,0x1f,%0\n\t"
251             "sll  %0,23,%2\n\t"         /* shift input to TEM bit location */
232             "and  %0,0x1f,%0\n\t"
233             "sll  %0,23,%%o1\n\t"      /* input to TEM bit location */
252             ".volatile\n\t"
253             "st   %%fsr,%1\n\t"
254             "ld   %1,%0\n\t"            /* %0 = fsr */
235             "st   %%fsr,%2\n\t"
236             "ld   %2,%0\n\t"           /* o0 = fsr */
255             "set  0x0f800000,%%o4\n\t"  /* mask of TEM (Trap Enable Mode bits) *
256             "andn %0,%%o4,%3\n\t"
257             "or   %2,%3,%2\n\t"         /* %2 = new fsr */
258             "st   %2,%1\n\t"
259             "ld   %1,%%fsr\n\t"
238             "andn %0,%%o4,%%o2\n\t"
239             "or   %%o1,%%o2,%%o1\n\t"  /* o1 = new fsr */
240             "st   %%o1,%2\n\t"
241             "ld   %2,%%fsr\n\t"
260             "srl  %0,23,%0\n\t"
261             "and  %0,0x1f,%0\n\t"
262             ".nonvolatile\n\t"
263             : "=r" (ret), "=m" (fsr), "=r" (tmp1), "=r" (tmp2)
264             : "r" (i)
265             : "cc");
245             : "=r" (ret)
246             : "0" (i), "m" (fsr)
247             : "o1", "o2", "o4");

267         return (ret);
268 }

270 extern __inline__ double
271 sqrt(double d)
272 {
273         return (__inline_sqrt(d));
255     double ret;
```



```
257     __asm__ __volatile__("fsqrtd %0,%0\n\t" : "=f" (ret) : "0" (d));
258     return (ret);
274 }

276 extern __inline__ float
277 sqrtf(float f)
278 {
279         return (__inline_sqrtf(f));
264     float ret;

266     __asm__ __volatile__("fsqrts %0,%0\n\t" : "=f" (ret) : "0" (f));
267     return (ret);
280 }

282 extern __inline__ double
283 fabs(double d)
284 {
285         double ret;

287         __asm__ __volatile__("fabsd %1,%0\n\t" : "=e" (ret) : "e" (d));
275     __asm__ __volatile__("fabsd %0,%0\n\t" : "=e" (ret) : "0" (d));
288         return (ret);
289 }

291 extern __inline__ float
292 fabsf(float f)
293 {
294         float ret;

296         __asm__ __volatile__("fabss %1,%0\n\t" : "=f" (ret) : "f" (f));
284     __asm__ __volatile__("fabss %0,%0\n\t" : "=f" (ret) : "0" (f));
297         return (ret);
298 }
______unchanged_portion_omitted_
```

```
**********************************************************
   32674 Sun May  4 03:07:06 2014
new/usr/src/lib/libm/sparc/src/locallibm.il
**********************************************************
```

```
   1 !
   2 ! CDDL HEADER START
   3 !
   4 ! The contents of this file are subject to the terms of the
   5 ! Common Development and Distribution License (the "License").
   6 ! You may not use this file except in compliance with the License.
   7 !
   8 ! You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
   9 ! or http://www.opensolaris.org/os/licensing.
  10 ! See the License for the specific language governing permissions
  11 ! and limitations under the License.
  12 !
  13 ! When distributing Covered Code,  this CDDL HEADER in each
  14 ! file and  the License file at usr/src/OPENSOLARIS.LICENSE.
  15 ! If applicable, add the following below this CDDL HEADER, with the
  16 ! fields enclosed by brackets "[]" replaced with your own identifying
  17 ! information: Portions Copyright [yyyy] [name of copyright owner]
  18 !
  19 ! CDDL HEADER END
  20 !
  21 ! Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
  22 !
  23 ! Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
  24 ! Use is subject to license terms.
  25 !

  27 ! Portions of this file are duplicated as GCC inline assembly in
  28 ! libm_inlines.h.  Keep them in sync.

  30         .inline __r_hypot_,2
  31         ld      [%o0],%o4
  32         sethi   0x1fffff,%o5
  33         or      %o5,1023,%o5
  34         and     %o4,%o5,%o4
  35         sethi   0x1fe000,%o3
  36         cmp     %o4,%o3
  37         ld      [%o0],%f0       ! load result with first argument
  38         bne     2f
  39         nop
  40         fabss   %f0,%f0
  41         ld      [%o1],%f1
  42         .volatile
  43         fcmps   %f0,%f1         ! generate invalid for Snan
  44         .nonvolatile
  45         nop
  46         fba     5f
  47         nop
  48 2:
  49         ld      [%o1],%o4
  50         sethi   0x1fffff,%o5
  51         or      %o5,1023,%o5
  52         and     %o4,%o5,%o4
  53         sethi   0x1fe000,%o3
  54         cmp     %o4,%o3
  55         bne     4f
  56         nop
  57         ld      [%o1],%f0       ! second argument inf
  58         fabss   %f0,%f0
  59         ld      [%o0],%f1
  60         .volatile
  61         fcmps   %f0,%f1         ! generate invalid for Snan
  62         .nonvolatile
```



```
  63         nop
  64         fba     5f
  65         nop
  66 4:
  67         ld      [%o1],%f3
  68         fsmuld  %f0,%f0,%f0
  69         fsmuld  %f3,%f3,%f2
  70         faddd   %f2,%f0,%f0
  71         fsqrtd  %f0,%f0
  72         fdtos   %f0,%f0
  73 5:
  74         .end

  76         .inline __c_abs,1
  77         ld      [%o0],%o4
  78         sethi   0x1fffff,%o5
  79         or      %o5,1023,%o5
  80         and     %o4,%o5,%o4
  81         sethi   0x1fe000,%o3
  82         cmp     %o4,%o3
  83         ld      [%o0],%f0
  84         bne     2f
  85         nop
  86         fabss   %f0,%f0
  87         ld      [%o0+4],%f1
  88         .volatile
  89         fcmps   %f0,%f1         ! generate invalid for Snan
  90         .nonvolatile
  91         nop
  92         fba     5f
  93         nop
  94 2:
  95         ld      [%o0+4],%o4
  96         sethi   0x1fffff,%o5
  97         or      %o5,1023,%o5
  98         and     %o4,%o5,%o4
  99         sethi   0x1fe000,%o3
 100         cmp     %o4,%o3
 101         bne     4f
 102         nop
 103         ld      [%o0+4],%f0
 104         fabss   %f0,%f0
 105         ld      [%o0],%f1
 106         .volatile
 107         fcmps   %f0,%f1         ! generate invalid for Snan
 108         .nonvolatile
 109         nop
 110         fba     5f
 111         nop
 112 ! store to 8-aligned address
 113 4:
 114         ld      [%o0+4],%f3
 115         fsmuld  %f0,%f0,%f0
 116         fsmuld  %f3,%f3,%f2
 117         faddd   %f2,%f0,%f0
 118         fsqrtd  %f0,%f0
 119         fdtos   %f0,%f0
 120 5:
 121         .end
 122 !- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 123 ! void
 124 ! __Fc_mult(c, a, b)
 125 ! complex *c, *a, *b;
 126 ! {

 128         .inline __Fc_mult,3
```

```
129 !    21         c->real = (a->real *  b->real) - (a->imag *  b->imag)
130         ld      [%o1+4],%f0     ! f0 = a->imag
131         ld      [%o2+4],%f1     ! f1 = b->imag
132         ld      [%o1],%f2       ! f2 = a->real
133         fsmuld   %f0,%f1,%f4     ! f4 = (a->imag *  b->imag)
134         ld      [%o2],%f3       ! f3 = b->real
135         fsmuld   %f2,%f1,%f6    ! f6 = a->real * b->imag
136         fsmuld   %f2,%f3,%f8    ! f8 =  a->real * b->real
137         fsmuld   %f0,%f3,%f10   ! f10 = a->imag * b->real
138         fsubd   %f8,%f4,%f0     ! f0 =  ar*br - ai*bi
139         faddd   %f6,%f10,%f2    ! f2 = ai*br + ar*bi
140         fdtos   %f0,%f4
141         fdtos   %f2,%f6
142         st      %f4,[%o0]
143         st      %f6,[%o0+4]
144         .end
145 ! }
146 !- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
147 ! void
148 ! __Fc_div(c, a, b)
149 ! complex *c, *a, *b;
150 ! {
151         .inline __Fc_div,3
152         ld      [%o2+4],%o3
153         sethi   %hi(0x7fffffff),%o4
154         or      %o4,%lo(0x7fffffff),%o4 ! [internal]
155         andcc   %o3,%o4,%g0
156         ld      [%o2],%f6               ! f6 gets reb
157         bne     1f
158         nop
159         ld      [%o1],%f0
160         ld      [%o2],%f1
161         fdivs   %f0,%f1,%f0
162         st      %f0,[%o0]
163         ld      [%o1+4],%f3
164         fdivs   %f3,%f1,%f3
165         st      %f3,[%o0+4]
166         ba      2f
167         nop
168 1:                                      ! [internal]
169         sethi   %hi(0x3ff00000),%o4
170         or      %g0,0,%o5
171         std     %o4,[%sp+0x48]
172         ldd     [%sp+0x48],%f8
173         ld      [%o2+4],%f10            ! f10 gets imb
174         fsmuld  %f6,%f6,%f16            ! f16/17 gets reb**2
175         ld      [%o1+4],%f4             ! f4 gets ima
176         fsmuld  %f10,%f10,%f12          ! f12/13 gets imb**2
177         ld      [%o1],%f19              ! f19 gets rea
178         fsmuld  %f4,%f10,%f0            ! f0/f1 gets ima*imb
179         fsmuld  %f19,%f6,%f2            ! f2/3 gets rea*reb
180         faddd   %f12,%f16,%f12          ! f12/13 gets reb**2+imb**2
181         fdivd   %f8,%f12,%f12           ! f12/13 gets 1/(reb**2+imb**2)
182         faddd   %f2,%f0,%f2             ! f2/3 gets rea*reb+ima*imb
183         fsmuld  %f4,%f6,%f24            ! f24/5 gets ima*reb
184         fmuld   %f2,%f12,%f2            ! f2/3 gets rec
185         fsmuld  %f19,%f10,%f10          ! f10/11 gets rea*imb
186         fsubd   %f24,%f10,%f10          ! f10/11 gets ima*reb-rea*imb
187         fmuld   %f10,%f12,%f12          ! f12 gets imc
188         fdtos   %f2,%f7                 ! f7 gets rec
189         fdtos   %f12,%f15               ! f15 gets imc
190         st      %f7,[%o0]
191         st      %f15,[%o0+4]
192 2:
193         .end
194 ! }
```



```
196         .inline .mul,2
197         .volatile
198         smul    %o0,%o1,%o0
199         rd      %y,%o1
200         sra     %o0,31,%o2
201         cmp     %o1,%o2
202         .nonvolatile
203         .end

205         .inline .umul,2
206         .volatile
207         umul    %o0,%o1,%o0
208         rd      %y,%o1
209         tst     %o1
210         .nonvolatile
211         .end

213         .inline .div,2
214         sra     %o0,31,%o4      ! extend sign
215         .volatile
216         wr      %o4,%g0,%y
217         cmp     %o1,0xffffffff  ! is divisor -1?
218         be,a    1f              ! if yes
219         .volatile
220         subcc   %g0,%o0,%o0     ! simply negate dividend
221         nop                     ! RT620 FABs A.0/A.1
222         sdiv    %o0,%o1,%o0     ! o0 contains quotient a/b
223         .nonvolatile
224 1:
225         .end

227         .inline .udiv,2
228         .volatile
229         wr      %g0,%g0,%y
230         nop
231         nop
232         nop
233         udiv    %o0,%o1,%o0     ! o0 contains quotient a/b
234         .nonvolatile
235         .end

237         .inline .rem,2
238         sra     %o0,31,%o4      ! extend sign
239         .volatile
240         wr      %o4,%g0,%y
241         cmp     %o1,0xffffffff  ! is divisor -1?
242         be,a    1f              ! if yes
243         .volatile
244         or      %g0,%g0,%o0     ! simply return 0
245         nop                     ! RT620 FABs A.0/A.1
246         sdiv    %o0,%o1,%o2     ! o2 contains quotient a/b
247         .nonvolatile
248         smul    %o2,%o1,%o4     ! o4 contains q*b
249         sub     %o0,%o4,%o0     ! o0 gets a-q*b
250 1:
251         .end

253         .inline .urem,2
254         .volatile
255         wr      %g0,%g0,%y
256         nop
257         nop
258         nop
259         udiv    %o0,%o1,%o2     ! o2 contains quotient a/b
260         .nonvolatile
```

```
261         umul    %o2,%o1,%o4     ! o4 contains q*b
262         sub     %o0,%o4,%o0     ! o0 gets a-q*b
263         .end

265         .inline .div_o3,2
266         sra     %o0,31,%o4      ! extend sign
267         .volatile
268         wr      %o4,%g0,%y
269         cmp     %o1,0xffffffff  ! is divisor -1?
270         be,a    1f              ! if yes
271         .volatile
272         subcc   %g0,%o0,%o0     ! simply negate dividend
273         mov     %o0,%o3         ! o3 gets __remainder
274         sdiv    %o0,%o1,%o0     ! o0 contains quotient a/b
275         .nonvolatile
276         smul    %o0,%o1,%o4     ! o4 contains q*b
277         ba      2f
278         sub     %o3,%o4,%o3     ! o3 gets a-q*b
279 1:
280         mov     %g0,%o3         ! __remainder is 0
281 2:
282         .end

284         .inline .udiv_o3,2
285         .volatile
286         wr      %g0,%g0,%y
287         mov     %o0,%o3         ! o3 gets __remainder
288         nop
289         nop
290         udiv    %o0,%o1,%o0     ! o0 contains quotient a/b
291         .nonvolatile
292         umul    %o0,%o1,%o4     ! o4 contains q*b
293         sub     %o3,%o4,%o3     ! o3 gets a-q*b
294         .end

296         .inline __ieee754_sqrt,2
297         std     %o0,[%sp+0x48]          ! store to 8-aligned address
298         ldd     [%sp+0x48],%f0
299         fsqrtd  %f0,%f0
300         .end

302         .inline __inline_sqrtf,1
303         st      %o0,[%sp+0x44]
304         ld      [%sp+0x44],%f0
305         fsqrts  %f0,%f0
306         .end

308         .inline __inline_sqrt,2
309         std     %o0,[%sp+0x48]          ! store to 8-aligned address
310         ldd     [%sp+0x48],%f0
311         fsqrtd  %f0,%f0
312         .end

314         .inline __sqrtf,1
315         st      %o0,[%sp+0x44]
316         ld      [%sp+0x44],%f0
317         fsqrts  %f0,%f0
318         .end

320         .inline __sqrt,2
321         std     %o0,[%sp+0x48]          ! store to 8-aligned address
322         ldd     [%sp+0x48],%f0
323         fsqrtd  %f0,%f0
324         .end

326         .inline __r_sqrt_,1
```



```
327         ld      [%o0],%f0
328         fsqrts  %f0,%f0
329         .end

331         .inline __d_sqrt_,1
332         ld      [%o0],%f0
333         ld      [%o0+4],%f1
334         fsqrtd  %f0,%f0
335         .end

337         .inline __ceil,2
338         std     %o0,[%sp+0x48]
339         sethi   %hi(0x80000000),%o5
340         andn    %o0,%o5,%o2
341         sethi   %hi(0x43300000),%o3
342         st      %g0,[%sp+0x54]
343         subcc   %o2,%o3,%g0
344         bl      1f
345         nop
346         sethi   %hi(0x3ff00000),%o2
347         st      %o2,[%sp+0x50]
348         ldd     [%sp+0x48],%f0
349         ldd     [%sp+0x50],%f2
350         fmuld   %f0,%f2,%f0
351         ba      4f
352         nop
353 1:
354         tst     %o0
355         st      %o3,[%sp+0x50]
356         ldd     [%sp+0x50],%f2
357         bge     2f
358         nop
359         fnegs   %f2,%f2
360 2:
361         ldd     [%sp+0x48],%f4
362         faddd   %f4,%f2,%f0
363         fsubd   %f0,%f2,%f0
364         fcmpd   %f0,%f4
365         sethi   %hi(0x3ff00000),%o2
366         st      %o2,[%sp+0x50]
367         and     %o0,%o5,%o4
368         fbge    3f
369         nop
370         ldd     [%sp+0x50],%f4
371         faddd   %f0,%f4,%f0
372 3:
373         st      %f0,[%sp+0x48]
374         ld      [%sp+0x48],%o3
375         andn    %o3,%o5,%o3
376         or      %o4,%o3,%o3
377         st      %o3,[%sp+0x48]
378         ld      [%sp+0x48],%f0
379 4:
380         .end

382         .inline __floor,2
383         std     %o0,[%sp+0x48]
384         sethi   %hi(0x80000000),%o5
385         andn    %o0,%o5,%o2
386         sethi   %hi(0x43300000),%o3
387         st      %g0,[%sp+0x54]
388         subcc   %o2,%o3,%g0
389         bl      1f
390         nop
391         sethi   %hi(0x3ff00000),%o2
392         st      %o2,[%sp+0x50]
```

```
393         ldd     [%sp+0x48],%f0
394         ldd     [%sp+0x50],%f2
395         fmuld   %f0,%f2,%f0
396         ba      4f
397         nop
398 1:
399         tst     %o0
400         st      %o3,[%sp+0x50]
401         ldd     [%sp+0x50],%f2
402         bge     2f
403         nop
404         fnegs   %f2,%f2
405 2:
406         ldd     [%sp+0x48],%f4
407         faddd   %f4,%f2,%f0
408         fsubd   %f0,%f2,%f0
409         fcmpd   %f0,%f4
410         sethi   %hi(0x3ff00000),%o2
411         st      %o2,[%sp+0x50]
412         ldd     [%sp+0x50],%f4
413         and     %o0,%o5,%o4
414         fble    3f
415         nop
416         fsubd   %f0,%f4,%f0
417 3:
418         st      %f0,[%sp+0x48]
419         ld      [%sp+0x48],%o3
420         andn    %o3,%o5,%o3
421         or      %o4,%o3,%o3
422         st      %o3,[%sp+0x48]
423         ld      [%sp+0x48],%f0
424 4:
425         .end

427         .inline __ilogb,2
428         sethi   %hi(0x7ff00000),%o4
429         andcc   %o4,%o0,%o2
430         bne     1f
431         nop
432         sethi   %hi(0x43500000),%o3
433         std     %o0,[%sp+0x48]
434         st      %o3,[%sp+0x50]
435         st      %g0,[%sp+0x54]
436         ldd     [%sp+0x48],%f0
437         ldd     [%sp+0x50],%f2
438         fmuld   %f0,%f2,%f0
439         sethi   %hi(0x80000001),%o0
440         or      %o0,%lo(0x80000001),%o0
441         st      %f0,[%sp+0x48]
442         ld      [%sp+0x48],%o2
443         andcc   %o2,%o4,%o2
444         srl     %o2,20,%o2
445         be      2f
446         nop
447         sub     %o2,0x435,%o0
448         ba      2f
449         nop
450 1:
451         subcc   %o4,%o2,%g0
452         srl     %o2,20,%o3
453         bne     0f
454         nop
455         sethi   %hi(0x7fffffff),%o0
456         or      %o0,%lo(0x7fffffff),%o0
457         ba      2f
458         nop
```



```
459 0:
460         sub     %o3,0x3ff,%o0
461 2:
462         .end

464         .inline __rint,2
465         std     %o0,[%sp+0x48]
466         sethi   %hi(0x80000000),%o2
467         andn    %o0,%o2,%o2
468         ldd     [%sp+0x48],%f0
469         sethi   %hi(0x43300000),%o3
470         st      %g0,[%sp+0x50]
471         st      %g0,[%sp+0x54]
472         subcc   %o2,%o3,%g0
473         bl      1f
474         nop
475         sethi   %hi(0x3ff00000),%o2
476         st      %o2,[%sp+0x50]
477         ldd     [%sp+0x50],%f2
478         fmuld   %f0,%f2,%f0
479         ba      3f
480         nop
481 1:
482         tst     %o0
483         st      %o3,[%sp+0x48]
484         st      %g0,[%sp+0x4c]
485         ldd     [%sp+0x48],%f2
486         bge     2f
487         nop
488         fnegs   %f2,%f2
489 2:
490         faddd   %f0,%f2,%f0
491         fcmpd   %f0,%f2
492         fbne    0f
493         nop
494         ldd     [%sp+0x50],%f0
495         bge     3f
496         nop
497         fnegs   %f0,%f0
498         ba      3f
499         nop
500 0:
501         fsubd   %f0,%f2,%f0
502 3:
503         .end

505         .inline __rintf,1
506         st      %o0,[%sp+0x48]
507         sethi   %hi(0x80000000),%o2
508         andn    %o0,%o2,%o2
509         ld      [%sp+0x48],%f0
510         sethi   %hi(0x4b000000),%o3
511         st      %g0,[%sp+0x50]
512         subcc   %o2,%o3,%g0
513         bl      1f
514         nop
515         sethi   %hi(0x3f800000),%o2
516         st      %o2,[%sp+0x50]
517         ld      [%sp+0x50],%f2
518         fmuls   %f0,%f2,%f0
519         ba      3f
520         nop
521 1:
522         tst     %o0
523         st      %o3,[%sp+0x48]
524         ld      [%sp+0x48],%f2
```

```
525         bge     2f
526         nop
527         fnegs   %f2,%f2
528 2:
529         fadds   %f0,%f2,%f0
530         fcmps   %f0,%f2
531         fbne    0f
532         nop
533         ld      [%sp+0x50],%f0
534         bge     3f
535         nop
536         fnegs   %f0,%f0
537         ba      3f
538         nop
539 0:
540         fsubs   %f0,%f2,%f0
541 3:
542         .end

544         .inline __min_subnormal,0
545         set     0x0,%o0
546         st      %o0,[%sp+0x44]
547         ld      [%sp+0x44],%f0
548         set     0x1,%o0
549         st      %o0,[%sp+0x44]
550         ld      [%sp+0x44],%f1
551         .end

553         .inline __d_min_subnormal_,0
554         set     0x0,%o0
555         st      %o0,[%sp+0x44]
556         ld      [%sp+0x44],%f0
557         set     0x1,%o0
558         st      %o0,[%sp+0x44]
559         ld      [%sp+0x44],%f1
560         .end

562         .inline __min_subnormalf,0
563         set     0x1,%o0
564         st      %o0,[%sp+0x44]
565         ld      [%sp+0x44],%f0
566         .end

568         .inline __r_min_subnormal_,0
569         set     0x1,%o0
570         st      %o0,[%sp+0x44]
571         ld      [%sp+0x44],%f0
572         .end

574         .inline __max_subnormal,0
575         set     0x000fffff,%o0
576         st      %o0,[%sp+0x44]
577         ld      [%sp+0x44],%f0
578         set     0xffffffff,%o0
579         st      %o0,[%sp+0x44]
580         ld      [%sp+0x44],%f1
581         .end

583         .inline __d_max_subnormal_,0
584         set     0x000fffff,%o0
585         st      %o0,[%sp+0x44]
586         ld      [%sp+0x44],%f0
587         set     0xffffffff,%o0
588         st      %o0,[%sp+0x44]
589         ld      [%sp+0x44],%f1
590         .end
```



```
592         .inline __max_subnormalf,0
593         set     0x007fffff,%o0
594         st      %o0,[%sp+0x44]
595         ld      [%sp+0x44],%f0
596         .end

598         .inline __r_max_subnormal_,0
599         set     0x007fffff,%o0
600         st      %o0,[%sp+0x44]
601         ld      [%sp+0x44],%f0
602         .end

604         .inline __min_normal,0
605         set     0x00100000,%o0
606         set     0x0,%o1
607         std     %o0,[%sp+0x48]
608         ldd     [%sp+0x48],%f0
609         .end

611         .inline __d_min_normal_,0
612         set     0x00100000,%o0
613         st      %o0,[%sp+0x44]
614         ld      [%sp+0x44],%f0
615         set     0x0,%o0
616         st      %o0,[%sp+0x44]
617         ld      [%sp+0x44],%f1
618         .end

620         .inline __min_normalf,0
621         set     0x00800000,%o0
622         st      %o0,[%sp+0x44]
623         ld      [%sp+0x44],%f0
624         .end

626         .inline __r_min_normal_,0
627         set     0x00800000,%o0
628         st      %o0,[%sp+0x44]
629         ld      [%sp+0x44],%f0
630         .end

632         .inline __max_normal,0
633         set     0x7fefffff,%o0
634         set     0xffffffff,%o1
635         std     %o0,[%sp+0x48]
636         ldd     [%sp+0x48],%f0
637         .end

639         .inline __d_max_normal_,0
640         set     0x7fefffff,%o0
641         st      %o0,[%sp+0x44]
642         ld      [%sp+0x44],%f0
643         set     0xffffffff,%o0
644         st      %o0,[%sp+0x44]
645         ld      [%sp+0x44],%f1
646         .end

648         .inline __max_normalf,0
649         set     0x7f7fffff,%o0
650         st      %o0,[%sp+0x44]
651         ld      [%sp+0x44],%f0
652         .end

654         .inline __r_max_normal_,0
655         set     0x7f7fffff,%o0
656         st      %o0,[%sp+0x44]
```

```
657         ld      [%sp+0x44],%f0
658         .end

660         .inline __infinity,0
661         set      0x7ff00000,%o0
662         set      0x0,%o1
663         std      %o0,[%sp+0x48]
664         ldd      [%sp+0x48],%f0
665         .end

667         .inline __infinity,0
668         set      0x7ff00000,%o0
669         set      0x0,%o1
670         std      %o0,[%sp+0x48]
671         ldd      [%sp+0x48],%f0
672         .end

674         .inline __d_infinity_,0
675         set     0x7ff00000,%o0
676         st      %o0,[%sp+0x44]
677         ld      [%sp+0x44],%f0
678         set     0x0,%o0
679         st      %o0,[%sp+0x44]
680         ld      [%sp+0x44],%f1
681         .end

683         .inline __infinityf,0
684         set     0x7f800000,%o0
685         st      %o0,[%sp+0x44]
686         ld      [%sp+0x44],%f0
687         .end

689         .inline __r_infinity_,0
690         set     0x7f800000,%o0
691         st      %o0,[%sp+0x44]
692         ld      [%sp+0x44],%f0
693         .end

695         .inline __signaling_nan,0
696         set     0x7ff00000,%o0
697         set     0x1,%o1
698         std     %o0,[%sp+0x48]
699         ldd     [%sp+0x48],%f0
700         .end

702         .inline __d_signaling_nan_,0
703         set     0x7ff00000,%o0
704         st      %o0,[%sp+0x44]
705         ld      [%sp+0x44],%f0
706         set     0x1,%o0
707         st      %o0,[%sp+0x44]
708         ld      [%sp+0x44],%f1
709         .end

711         .inline __signaling_nanf,0
712         set     0x7f800001,%o0
713         st      %o0,[%sp+0x44]
714         ld      [%sp+0x44],%f0
715         .end

717         .inline __r_signaling_nan_,0
718         set     0x7f800001,%o0
719         st      %o0,[%sp+0x44]
720         ld      [%sp+0x44],%f0
721         .end
```



```
723         .inline __quiet_nan,0
724         set     0x7fffffff,%o0
725         st      %o0,[%sp+0x44]
726         ld      [%sp+0x44],%f0
727         set     0xffffffff,%o0
728         st      %o0,[%sp+0x44]
729         ld      [%sp+0x44],%f1
730         .end

732         .inline __d_quiet_nan_,0
733         set     0x7fffffff,%o0
734         st      %o0,[%sp+0x44]
735         ld      [%sp+0x44],%f0
736         set     0xffffffff,%o0
737         st      %o0,[%sp+0x44]
738         ld      [%sp+0x44],%f1
739         .end

741         .inline __quiet_nanf,0
742         set     0x7fffffff,%o0
743         st      %o0,[%sp+0x44]
744         ld      [%sp+0x44],%f0
745         .end

747         .inline __r_quiet_nan_,0
748         set     0x7fffffff,%o0
749         st      %o0,[%sp+0x44]
750         ld      [%sp+0x44],%f0
751         .end

753         .inline __swapEX,1
754         and     %o0,0x1f,%o1
755         sll     %o1,5,%o1               ! shift input to aexc bit location
755         sll     %o1,5,%o1               !  input to aexc bit location
756         .volatile
757         st      %fsr,[%sp+0x44]
758         ld      [%sp+0x44],%o0          ! o0 = fsr
759         andn    %o0,0x3e0,%o2
760         or      %o1,%o2,%o1             ! o1 = new fsr
761         st      %o1,[%sp+0x44]
762         ld      [%sp+0x44],%fsr
763         srl     %o0,5,%o0
764         and     %o0,0x1f,%o0
765         .nonvolatile
766         .end

768         .inline _QgetRD,0
769         st      %fsr,[%sp+0x44]
770         ld      [%sp+0x44],%o0          ! o0 = fsr
771         srl     %o0,30,%o0              ! return __round control value
772         .end

774         .inline _QgetRP,0
775         or      %g0,%g0,%o0
776         .end

778         .inline __swapRD,1
779         and     %o0,0x3,%o0
780         sll     %o0,30,%o1              ! shift input to RD bit location
780         sll     %o0,30,%o1              !  input to RD bit location
781         .volatile
782         st      %fsr,[%sp+0x44]
783         ld      [%sp+0x44],%o0          ! o0 = fsr
784         set     0xc0000000,%o4          ! mask of rounding direction bits
785         andn    %o0,%o4,%o2
786         or      %o1,%o2,%o1             ! o1 = new fsr
```

```
787         st      %o1,[%sp+0x44]
788         ld      [%sp+0x44],%fsr
789         srl     %o0,30,%o0
790         and     %o0,0x3,%o0
791         .nonvolatile
792         .end
793 !
794 ! On the SPARC, __swapRP is a no-op; always return 0 for backward compatibility
795 !

797         .inline __swapRP,1
798         or      %g0,%g0,%o0
799         .end

801         .inline __swapTE,1
802         and     %o0,0x1f,%o0
803         sll     %o0,23,%o1              ! shift input to TEM bit location
803         sll     %o0,23,%o1              !  input to TEM bit location
804         .volatile
805         st      %fsr,[%sp+0x44]
806         ld      [%sp+0x44],%o0            ! o0 = fsr
807         set     0x0f800000,%o4          ! mask of TEM (Trap Enable Mode bits)
808         andn    %o0,%o4,%o2
809         or      %o1,%o2,%o1             ! o1 = new fsr
810         st      %o1,[%sp+0x48]
811         ld      [%sp+0x48],%fsr
812         srl     %o0,23,%o0
813         and     %o0,0x1f,%o0
814         .nonvolatile
815         .end

817         .inline __fp_class,2
818         sethi   %hi(0x80000000),%o2     ! o2 gets 80000000
819         andn    %o0,%o2,%o0             ! o0-o1 gets abs(x)
820         orcc    %o0,%o1,%g0             ! set cc as x is zero/nonzero
821         bne     1f                      ! branch if x is nonzero
822         nop
823         mov     0,%o0
824         ba      2f                      ! x is 0
825         nop
826 1:
827         sethi   %hi(0x7ff00000),%o2     ! o2 gets 7ff00000
828         andcc   %o0,%o2,%g0             ! cc set by __exp field of x
829         bne     1f                      ! branch if normal or max __exp
830         nop
831         mov     1,%o0
832         ba      2f                      ! x is subnormal
833         nop
834 1:
835         cmp     %o0,%o2
836         bge     1f                      ! branch if x is max __exp
837         nop
838         mov     2,%o0
839         ba      2f                      ! x is normal
840         nop
841 1:
842         andn    %o0,%o2,%o0             ! o0 gets msw __significand field
843         orcc    %o0,%o1,%g0             ! set cc by OR __significand
844         bne     1f                      ! Branch if __nan
845         nop
846         mov     3,%o0
847         ba      2f                      ! x is __infinity
848         nop
849 1:
850         sethi   %hi(0x00080000),%o2
851         andcc   %o0,%o2,%g0             ! set cc by quiet/sig bit
```



```
852         be      1f                      ! Branch if signaling
853         nop
854         mov     4,%o0                   ! x is quiet NaN
855         ba      2f
856         nop
857 1:
858         mov     5,%o0                   ! x is signaling NaN
859 2:
860         .end

862         .inline __fp_classf,1
863         sethi   %hi(0x80000000),%o2
864         andncc  %o0,%o2,%o0
865         bne     1f
866         nop
867         mov     0,%o0
868         ba      2f                      ! x is 0
869         nop
870 1:
871         sethi   %hi(0x7f800000),%o2
872         andcc   %o0,%o2,%g0
873         bne     1f
874         nop
875         mov     1,%o0
876         ba      2f                      ! x is subnormal
877         nop
878 1:
879         cmp     %o0,%o2
880         bge     1f
881         nop
882         mov     2,%o0
883         ba      2f                      ! x is normal
884         nop
885 1:
886         bg      1f
887         nop
888         mov     3,%o0
889         ba      2f                      ! x is __infinity
890         nop
891 1:
892         sethi   %hi(0x00400000),%o2
893         andcc   %o0,%o2,%g0
894         mov     4,%o0                   ! x is quiet NaN
895         bne     2f
896         nop
897         mov     5,%o0                   ! x is signaling NaN
898 2:
899         .end

901         .inline __ir_fp_class_,1
902         ld      [%o0],%o0
903         sethi   %hi(0x80000000),%o2
904         andncc  %o0,%o2,%o0
905         bne     1f
906         nop
907         mov     0,%o0
908         ba      2f                      ! x is 0
909         nop
910 1:
911         sethi   %hi(0x7f800000),%o2
912         andcc   %o0,%o2,%g0
913         bne     1f
914         nop
915         mov     1,%o0
916         ba      2f                      ! x is subnormal
917         nop
```

```
 918 1:
 919         cmp     %o0,%o2
 920         bge     1f
 921         nop
 922         mov     2,%o0
 923         ba      2f                      ! x is normal
 924         nop
 925 1:
 926         bg      1f
 927         nop
 928         mov     3,%o0
 929         ba      2f                      ! x is __infinity
 930         nop
 931 1:
 932         sethi   %hi(0x00400000),%o2
 933         andcc   %o0,%o2,%g0
 934         mov     4,%o0                   ! x is quiet NaN
 935         bne     2f
 936         nop
 937         mov     5,%o0                   ! x is signaling NaN
 938 2:
 939         .end

 941         .inline __copysign,4
 942         set     0x80000000,%o3
 943         and     %o2,%o3,%o2
 944         andn    %o0,%o3,%o0
 945         or      %o0,%o2,%o0
 946         std      %o0,[%sp+0x48]
 947         ldd     [%sp+0x48],%f0
 948         .end

 950         .inline __copysignf,2
 951         set     0x80000000,%o2
 952         andn    %o0,%o2,%o0
 953         and     %o1,%o2,%o1
 954         or      %o0,%o1,%o0
 955         st      %o0,[%sp+0x44]
 956         ld      [%sp+0x44],%f0
 957         .end

 959         .inline __r_copysign_,2
 960         ld      [%o0],%o0
 961         ld      [%o1],%o1
 962         set     0x80000000,%o2
 963         andn    %o0,%o2,%o0
 964         and     %o1,%o2,%o1
 965         or      %o0,%o1,%o0
 966         st      %o0,[%sp+0x44]
 967         ld      [%sp+0x44],%f0
 968         .end

 970         .inline _finite,2
 971         set     0x7ff00000,%o1
 972         and     %o0,%o1,%o0
 973         cmp     %o0,%o1
 974         mov     1,%o0
 975         bne     1f
 976         nop
 977         mov     0,%o0
 978 1:
 979         .end

 981         .inline __finitef,2
 982         set     0x7f800000,%o1
 983         and     %o0,%o1,%o0
```



```
 984         cmp     %o0,%o1
 985         mov     1,%o0
 986         bne     1f
 987         nop
 988         mov     0,%o0
 989 1:
 990         .end

 992         .inline __ir_finite_,1
 993         ld      [%o0],%o0
 994         set     0x7f800000,%o1
 995         and     %o0,%o1,%o0
 996         cmp     %o0,%o1
 997         mov     1,%o0
 998         bne     1f
 999         nop
1000         mov     0,%o0
1001 1:
1002         .end

1004         .inline __signbit,1
1005         srl     %o0,31,%o0
1006         .end

1008         .inline __signbitf,1
1009         srl     %o0,31,%o0
1010         .end

1012         .inline __ir_signbit_,1
1013         ld      [%o0],%o0
1014         srl     %o0,31,%o0
1015         .end

1017         .inline __isinf,2
1018         tst     %o1
1019         sethi   %hi(0x80000000),%o2
1020         bne     1f
1021         nop
1022         andn    %o0,%o2,%o0
1023         sethi   %hi(0x7ff00000),%o2
1024         cmp     %o0,%o2
1025         mov     1,%o0
1026         be      2f
1027         nop
1028 1:
1029         mov     0,%o0
1030 2:
1031         .end

1033         .inline __isinff,1
1034         sethi   %hi(0x80000000),%o2
1035         andn    %o0,%o2,%o0             ! o0 gets abs(x)
1036         sethi   %hi(0x7f800000),%o2
1037         cmp     %o0,%o2
1038         mov     0,%o0
1039         bne     1f                      ! Branch if not inf.
1040         nop
1041         mov     1,%o0
1042 1:
1043         .end

1045         .inline __ir_isinf_,1
1046         ld      [%o0],%o0
1047         sethi   %hi(0x80000000),%o2
1048         andn    %o0,%o2,%o0             ! o0 gets abs(x)
1049         sethi   %hi(0x7f800000),%o2
```

```
1050         cmp     %o0,%o2
1051         mov     0,%o0
1052         bne     1f                      ! Branch if not inf.
1053         nop
1054         mov     1,%o0
1055 1:
1056         .end

1058         .inline __isnan,2
1059         sethi   %hi(0x80000000),%o2
1060         andn    %o0,%o2,%o0
1061         sub     %g0,%o1,%o3
1062         or      %o1,%o3,%o1
1063         srl     %o1,31,%o1
1064         or      %o0,%o1,%o0
1065         sethi   %hi(0x7ff00000),%o4
1066         sub     %o4,%o0,%o0
1067         srl     %o0,31,%o0
1068         .end

1070         .inline __isnanf,1
1071         sethi   %hi(0x80000000),%o2
1072         andn    %o0,%o2,%o0
1073         sethi   %hi(0x7f800000),%o1
1074         sub     %o1,%o0,%o0
1075         srl     %o0,31,%o0
1076         .end

1078         .inline __ir_isnan_,1
1079         ld      [%o0],%o0
1080         sethi   %hi(0x80000000),%o2
1081         andn    %o0,%o2,%o0
1082         sethi   %hi(0x7f800000),%o1
1083         sub     %o1,%o0,%o0
1084         srl     %o0,31,%o0
1085         .end

1087         .inline __isnormal,2
1088         sethi   %hi(0x80000000),%o2
1089         andn    %o0,%o2,%o0
1090         sethi   %hi(0x7ff00000),%o2
1091         cmp     %o0,%o2
1092         sethi   %hi(0x00100000),%o2
1093         bge     1f
1094         nop
1095         cmp     %o0,%o2
1096         mov     1,%o0
1097         bge     2f
1098         nop
1099 1:
1100         mov     0,%o0
1101 2:
1102         .end

1104         .inline __isnormalf,1
1105         sethi   %hi(0x80000000),%o2
1106         andn    %o0,%o2,%o0
1107         sethi   %hi(0x7f800000),%o2
1108         cmp     %o0,%o2
1109         sethi   %hi(0x00800000),%o2
1110         bge     1f
1111         nop
1112         cmp     %o0,%o2
1113         mov     1,%o0
1114         bge     2f
1115         nop
```



```
1116 1:
1117         mov     0,%o0
1118 2:
1119         .end

1121         .inline __ir_isnormal_,1
1122         ld      [%o0],%o0
1123         sethi   %hi(0x80000000),%o2
1124         andn    %o0,%o2,%o0
1125         sethi   %hi(0x7f800000),%o2
1126         cmp     %o0,%o2
1127         sethi   %hi(0x00800000),%o2
1128         bge     1f
1129         nop
1130         cmp     %o0,%o2
1131         mov     1,%o0
1132         bge     2f
1133         nop
1134 1:
1135         mov     0,%o0
1136 2:
1137         .end

1139         .inline __issubnormal,2
1140         sethi   %hi(0x80000000),%o2     ! o2 gets 80000000
1141         andn    %o0,%o2,%o0             ! o0/o1 gets abs(x)
1142         sethi   %hi(0x00100000),%o2     ! o2 gets 00100000
1143         cmp     %o0,%o2
1144         bge     1f                      ! branch if x norm or max __exp
1145         nop
1146         orcc    %o0,%o1,%g0
1147         be      1f                      ! Branch if x zero
1148         nop
1149         mov     1,%o0                   ! x is subnormal
1150         ba      2f
1151         nop
1152 1:
1153         mov     0,%o0
1154 2:
1155         .end

1157         .inline __issubnormalf,1
1158         sethi   %hi(0x80000000),%o2     ! o2 gets 80000000
1159         andn    %o0,%o2,%o0             ! o0 gets abs(x)
1160         sethi   %hi(0x00800000),%o2     ! o2 gets 00800000
1161         cmp     %o0,%o2
1162         bge     1f                      ! branch if x norm or max __exp
1163         nop
1164         orcc    %o0,%g0,%g0
1165         be      1f                      ! Branch if x zero
1166         nop
1167         mov     1,%o0                   ! x is subnormal
1168         ba      2f
1169         nop
1170 1:
1171         mov     0,%o0
1172 2:
1173         .end

1175         .inline __ir_issubnormal_,1
1176         ld      [%o0],%o0
1177         sethi   %hi(0x80000000),%o2     ! o2 gets 80000000
1178         andn    %o0,%o2,%o0             ! o0 gets abs(x)
1179         sethi   %hi(0x00800000),%o2     ! o2 gets 00800000
1180         cmp     %o0,%o2
1181         bge     1f                      ! branch if x norm or max __exp
```

```
1182         nop
1183         orcc    %o0,%g0,%g0
1184         be      1f                      ! Branch if x zero
1185         nop
1186         mov     1,%o0                   ! x is subnormal
1187         ba      2f
1188         nop
1189 1:
1190         mov     0,%o0
1191 2:
1192         .end

1194         .inline __iszero,2
1195         sethi   %hi(0x80000000),%o2
1196         andn    %o0,%o2,%o0
1197         orcc    %o0,%o1,%g0
1198         mov     1,%o0
1199         be      1f
1200         nop
1201         mov     0,%o0
1202 1:
1203         .end

1205         .inline __iszerof,1
1206         sethi   %hi(0x80000000),%o2
1207         andncc  %o0,%o2,%o0
1208         mov     1,%o0
1209         be      1f
1210         nop
1211         mov     0,%o0
1212 1:
1213         .end

1215         .inline __ir_iszero_,1
1216         ld      [%o0],%o0
1217         sethi   %hi(0x80000000),%o2
1218         andncc  %o0,%o2,%o0
1219         mov     1,%o0
1220         be      1f
1221         nop
1222         mov     0,%o0
1223 1:
1224         .end

1226         .inline abs,1
1227         sra     %o0,31,%o1
1228         xor     %o0,%o1,%o0
1229         sub     %o0,%o1,%o0
1230         .end

1232         .inline __fabs,2
1233         st      %o0,[%sp+0x48]
1234         st      %o1,[%sp+0x4c]
1235         ldd     [%sp+0x48],%f0
1236         fabsd   %f0,%f0
1237         .end

1239         .inline __fabsf,1
1240         st      %o0,[%sp+0x44]
1241         ld      [%sp+0x44],%f0
1242         fabss   %f0,%f0
1243         .end

1245         .inline __r_fabs_,1
1246         ld      [%o0],%f0
1247         fabss   %f0,%f0
```



```
1248         .end
1249 !
1250 !       __nintf - f77 NINT(REAL*4)
1251 !

1253         .inline __nintf,1
1254         srl     %o0,30-7,%g1
1255         sethi   %hi(0x7fffff),%o2
1256         st      %o0,[%sp+0x44]
1257         and     %g1,0xff,%g1
1258         or      %o2,%lo(0x7fffff),%o2
1259         sethi   %hi(1<<22),%o4
1260         subcc   %g1,127+31,%g0
1261         and     %o0,%o2,%o3
1262         bl      0f
1263         nop
1264         sethi   %hi(0xcf000000),%o2
1265         sethi   %hi(0x80000000),%g1
1266         subcc   %o0,%o2,%g0
1267         or      %g1,%g0,%o0
1268         be      9f
1269         nop
1270         ld      [%sp+0x44],%f0
1271         fstoi   %f0,%f0
1272         st      %f0,[%sp+0x44]
1273         ld      [%sp+0x44],%o0
1274         ba      9f
1275         nop
1276 0:
1277         add     %o4,%o4,%o5
1278         or      %o3,%o5,%o3
1279         sra     %o0,31-0,%o2
1280         subcc   %g1,127,%g1
1281         srl     %o4,%g1,%o4
1282         bge     1f
1283         nop
1284         subcc   %g1,-1,%g0
1285         or      %g0,0,%o0
1286         bne     2f
1287         nop
1288         or      %g0,1,%o0
1289         ba      2f
1290         nop
1291 1:
1292         add     %o3,%o4,%o3
1293         or      %g0,23,%o0
1294         subcc   %o0,%g1,%o0
1295         bl      1f
1296         nop
1297         srl     %o3,%o0,%o0
1298         ba      2f
1299         nop
1300 1:
1301         sub     %g0,%o0,%o0
1302         sll     %o3,%o0,%o0
1303 2:
1304         xor     %o0,%o2,%o0
1305         and     %o2,1,%o2
1306         add     %o0,%o2,%o0
1307 9:
1308         .end

1310         .inline __il_nint,1
1311         ld      [%o0],%o0
1312         sra     %o0,0,%o0
1313         srlx    %o0,31-8,%g1
```

```
1314         or      %g0,1,%o2
1315         sllx    %o2,23-1,%o4
1316         and     %g1,0xff,%g1
1317         sllx    %o2,63-0,%o2
1318         subcc   %g1,127+63,%g0
1319         bl      0f
1320         nop
1321         st      %o0,[%sp+0x48]
1322         ld      [%sp+0x48],%f0
1323         fstox   %f0,%f0
1324         std     %f0,[%sp+0x48]
1325         ldx     [%sp+0x48],%o1
1326         ba      9f
1327         nop
1328 0:
1329         add     %o4,%o4,%o5
1330         srax    %o2,63-23,%o2
1331         sub     %g1,127+23,%o1
1332         xnor    %o2,%g0,%o2
1333         and     %o0,%o2,%o3
1334         or      %o3,%o5,%o3
1335         srax    %o0,63-0,%o2
1336         subcc   %g1,127,%g1
1337         bge     1f
1338         nop
1339         subcc   %g1,-1,%g0
1340         or      %g0,0,%o0
1341         bne     2f
1342         nop
1343         or      %g0,1,%o0
1344         ba      2f
1345         nop
1346 1:
1347         brlz,pt %o1,3f
1348         nop
1349         sub     %g1,23,%o0
1350         sllx    %o3,%o0,%o0
1351         ba      2f
1352         nop
1353 3:
1354         srlx    %o4,%g1,%o4
1355         add     %o3,%o4,%o3
1356         or      %g0,23,%o0
1357         sub     %o0,%g1,%o0
1358         srlx    %o3,%o0,%o0
1359 2:
1360         xor     %o0,%o2,%o0
1361         sub     %o0,%o2,%o1
1362 9:
1363         srlx    %o1,32,%o0
1364         .end
1365 !
1366 !       __i_dnnt - f77 NINT(REAL*8)
1367 !

1369         .inline __i_dnnt,1
1370         ld      [%o0],%o1
1371         sllx    %o1,32,%o1
1372         ld      [%o0+4],%o0
1373         or      %o0,%o1,%o0
1374         srlx    %o0,63-11,%g1
1375         or      %g0,1,%o2
1376         stx     %o0,[%sp+0x48]
1377         sllx    %o2,52-1,%o4
1378         and     %g1,0x7ff,%g1
1379         sllx    %o2,63-0,%o2
```



```
1380         subcc   %g1,1023+32,%g0
1381         bl      0f
1382         nop
1383         ldd     [%sp+0x48],%f0
1384         ba      8f
1385         nop
1386 0:
1387         add     %o4,%o4,%o5
1388         srax    %o2,63-52,%o2
1389         sub     %g1,1023+30,%o1
1390         xnor    %o2,%g0,%o2
1391         and     %o0,%o2,%o3
1392         or      %o3,%o5,%o3
1393         srax    %o0,63-0,%o2
1394         subcc   %g1,1023,%g1
1395         bge     1f
1396         nop
1397         subcc   %g1,-1,%g0
1398         or      %g0,0,%o0
1399         bne     2f
1400         nop
1401         or      %g0,1,%o0
1402         ba      2f
1403         nop
1404 1:
1405         srlx    %o4,%g1,%o4
1406         add     %o3,%o4,%o3
1407         or      %g0,52,%o0
1408         sub     %o0,%g1,%o0
1409         srlx    %o3,%o0,%o0
1410 2:
1411         xor     %o0,%o2,%o0
1412         sub     %o0,%o2,%o0
1413         brlz,pt %o1,9f
1414         nop
1415         stx     %o0,[%sp+0x48]
1416         ldd     [%sp+0x48],%f0
1417         fxtod   %f0,%f0
1418 8:
1419         fdtoi   %f0,%f0
1420         st      %f0,[%sp+0x44]
1421         ld      [%sp+0x44],%o0
1422 9:
1423         .end

1425         .inline __il_dnnt,1
1426         ld      [%o0],%o1
1427         sllx    %o1,32,%o1
1428         ld      [%o0+4],%o0
1429         or      %o0,%o1,%o0
1430         srlx    %o0,63-11,%g1
1431         or      %g0,1,%o2
1432         sllx    %o2,52-1,%o4
1433         and     %g1,0x7ff,%g1
1434         sllx    %o2,63-0,%o2
1435         subcc   %g1,1023+63,%g0
1436         bl      0f
1437         nop
1438         stx     %o0,[%sp+0x48]
1439         ldd     [%sp+0x48],%f0
1440         fdtox   %f0,%f0
1441         std     %f0,[%sp+0x48]
1442         ldx     [%sp+0x48],%o1
1443         ba      9f
1444         nop
1445 0:
```

```
1446         add     %o4,%o4,%o5
1447         srax    %o2,63-52,%o2
1448         sub     %g1,1023+52,%o1
1449         xnor    %o2,%g0,%o2
1450         and     %o0,%o2,%o3
1451         or      %o3,%o5,%o3
1452         srax    %o0,63-0,%o2
1453         subcc   %g1,1023,%g1
1454         bge     1f
1455         nop
1456         subcc   %g1,-1,%g0
1457         or      %g0,0,%o0
1458         bne     2f
1459         nop
1460         or      %g0,1,%o0
1461         ba      2f
1462         nop
1463 1:
1464         brlz,pt %o1,3f
1465         nop
1466         sub     %g1,52,%o0
1467         sllx    %o3,%o0,%o0
1468         ba      2f
1469         nop
1470 3:
1471         srlx    %o4,%g1,%o4
1472         add     %o3,%o4,%o3
1473         or      %g0,52,%o0
1474         sub     %o0,%g1,%o0
1475         srlx    %o3,%o0,%o0
1476 2:
1477         xor     %o0,%o2,%o0
1478         sub     %o0,%o2,%o1
1479 9:
1480         srlx    %o1,32,%o0
1481         .end

1483         .inline __anintf,1
1484         or      %g0,1,%o1
1485         srl     %o0,23,%g1
1486         and     %g1,0xff,%g1
1487         sub     %g0,%g1,%g1
1488         add     %g1,0x95,%g1
1489         subcc   %g1,23,%g0
1490         sll     %o1,%g1,%o1
1491         sub     %o1,1,%o2
1492         bcs     1f
1493         nop
1494         be      2f
1495         nop
1496         bl      3f
1497         nop
1498         sethi   %hi(0x80000000),%o1
1499         and     %o0,%o1,%o0
1500         ba      3f
1501         nop
1502 1:
1503         and     %o0,%o1,%o1
1504 2:
1505         add     %o0,%o1,%o0
1506         andn    %o0,%o2,%o0
1507 3:
1508         st      %o0,[%sp+0x48]
1509         ld      [%sp+0x48],%f0
1510         .end
```



```
1512         .inline __anint,2
1513         sllx    %o0,32,%o0
1514         or      %o0,%o1,%o0
1515         or      %g0,1,%o1
1516         srlx    %o0,52,%g1
1517         and     %g1,0x7ff,%g1
1518         sub     %g0,%g1,%g1
1519         add     %g1,0x432,%g1
1520         subcc   %g1,52,%g0
1521         sllx    %o1,%g1,%o1
1522         sub     %o1,1,%o2
1523         bcs,pt  %icc,1f
1524         nop
1525         be,pt   %icc,2f
1526         nop
1527         bl,pt   %icc,3f
1528         nop
1529         srlx    %o0,63,%o0
1530         sllx    %o0,63,%o0
1531         ba      3f
1532         nop
1533 1:
1534         and     %o0,%o1,%o1
1535 2:
1536         add     %o0,%o1,%o0
1537         andn    %o0,%o2,%o0
1538 3:
1539         stx     %o0,[%sp+0x48]
1540         ldd     [%sp+0x48],%f0
1541         .end

1543         .inline __Fz_minus,3
1544         ld      [%o1],%f0
1545         ld      [%o1+0x4],%f1
1546         ld      [%o2],%f4
1547         ld      [%o2+0x4],%f5
1548         fsubd   %f0,%f4,%f0
1549         ld      [%o1+8],%f2
1550         ld      [%o1+0xc],%f3
1551         ld      [%o2+8],%f6
1552         ld      [%o2+0xc],%f7
1553         fsubd   %f2,%f6,%f2
1554         st      %f0,[%o0+0x0]
1555         st      %f1,[%o0+0x4]
1556         st      %f2,[%o0+0x8]
1557         st      %f3,[%o0+0xc]
1558         .end

1560         .inline __Fz_add,3
1561         ld      [%o1],%f0
1562         ld      [%o1+0x4],%f1
1563         ld      [%o2],%f4
1564         ld      [%o2+0x4],%f5
1565         faddd   %f0,%f4,%f0
1566         ld      [%o1+8],%f2
1567         ld      [%o1+0xc],%f3
1568         ld      [%o2+8],%f6
1569         ld      [%o2+0xc],%f7
1570         faddd   %f2,%f6,%f2
1571         st      %f0,[%o0+0x0]
1572         st      %f1,[%o0+0x4]
1573         st      %f2,[%o0+0x8]
1574         st      %f3,[%o0+0xc]
1575         .end

1577         .inline __Fz_neg,2
```

```
1578         ld      [%o1],%f0
1579         fnegs   %f0,%f0
1580         ld      [%o1+0x4],%f1
1581         st      %f1,[%o0+0x4]
1582         ld      [%o1+8],%f2
1583         fnegs   %f2,%f2
1584         ld      [%o1+0xc],%f3
1585         st      %f3,[%o0+0xc]
1586         st      %f0,[%o0]
1587         st      %f2,[%o0+0x8]
1588         .end

1590         .inline __Ff_conv_z,2
1591         st      %o1,[%sp+0x44]
1592         ld      [%sp+0x44],%f0
1593         fstod   %f0,%f0
1594         st      %g0,[%o0+0x8]
1595         st      %g0,[%o0+0xc]
1596         st      %f1,[%o0+0x4]
1597         st      %f0,[%o0]
1598         .end

1600         .inline __Fz_conv_f,1
1601         ld      [%o0],%f0
1602         ld      [%o0+4],%f1
1603         fdtos   %f0,%f0
1604         .end

1606         .inline __Fz_conv_i,1
1607         ld      [%o0],%f0
1608         ld      [%o0+4],%f1
1609         fdtoi   %f0,%f0
1610         st      %f0,[%sp+0x44]
1611         ld      [%sp+0x44],%o0
1612         .end

1614         .inline __Fi_conv_z,2
1615         st      %o1,[%sp+0x44]
1616         ld      [%sp+0x44],%f0
1617         fitod   %f0,%f0
1618         st      %g0,[%o0+0x8]
1619         st      %g0,[%o0+0xc]
1620         st      %f1,[%o0+0x4]
1621         st      %f0,[%o0]
1622         .end

1624         .inline __Fz_conv_d,1
1625         ld      [%o0],%f0
1626         ld      [%o0+4],%f1
1627         .end

1629         .inline __Fd_conv_z,3
1630         st      %o1,[%o0]
1631         st      %o2,[%o0+0x4]
1632         st      %g0,[%o0+0x8]
1633         st      %g0,[%o0+0xc]
1634         .end

1636         .inline __Fz_conv_c,2
1637         ldd     [%o1],%f0
1638         fdtos   %f0,%f0
1639         st      %f0,[%o0]
1640         ldd     [%o1+0x8],%f2
1641         fdtos   %f2,%f1
1642         st      %f1,[%o0+0x4]
1643         .end
```



```
1645         .inline __Fz_eq,2
1646         ld      [%o0],%f0
1647         ld      [%o0+4],%f1
1648         ld      [%o1],%f2
1649         ld      [%o1+4],%f3
1650         fcmpd   %f0,%f2
1651         mov     %o0,%o2
1652         mov     0,%o0
1653         fbne    1f
1654         nop
1655         ld      [%o2+8],%f0
1656         ld      [%o2+12],%f1
1657         ld      [%o1+8],%f2
1658         ld      [%o1+12],%f3
1659         fcmpd   %f0,%f2
1660         nop
1661         fbne    1f
1662         nop
1663         mov     1,%o0
1664 1:
1665         .end

1667         .inline __Fz_ne,2
1668         ld      [%o0],%f0
1669         ld      [%o0+4],%f1
1670         ld      [%o1],%f2
1671         ld      [%o1+4],%f3
1672         fcmpd   %f0,%f2
1673         mov     %o0,%o2
1674         mov     1,%o0
1675         fbne    1f
1676         nop
1677         ld      [%o2+8],%f0
1678         ld      [%o2+12],%f1
1679         ld      [%o1+8],%f2
1680         ld      [%o1+12],%f3
1681         fcmpd   %f0,%f2
1682         nop
1683         fbne    1f
1684         nop
1685         mov     0,%o0
1686 1:
1687         .end

1689         .inline __c_cmplx,3
1690         ld      [%o1],%o1
1691         st      %o1,[%o0]
1692         ld      [%o2],%o2
1693         st      %o2,[%o0+4]
1694         .end

1696         .inline __d_cmplx,3
1697         ld      [%o1],%f0
1698         st      %f0,[%o0]
1699         ld      [%o1+4],%f1
1700         st      %f1,[%o0+4]
1701         ld      [%o2],%f0
1702         st      %f0,[%o0+0x8]
1703         ld      [%o2+4],%f1
1704         st      %f1,[%o0+0xc]
1705         .end

1707         .inline __r_cnjg,2
1708         ld      [%o1+0x4],%f1
1709         fnegs   %f1,%f1
```

```
1710         ld      [%o1],%f0
1711         st      %f0,[%o0]
1712         st      %f1,[%o0+4]
1713         .end

1715         .inline __d_cnjg,2
1716         ld      [%o1+0x8],%f0
1717         fnegs   %f0,%f0
1718         ld      [%o1+0xc],%f1
1719         st      %f1,[%o0+0xc]
1720         ld      [%o1+0x0],%f1
1721         st      %f1,[%o0+0x0]
1722         ld      [%o1+0x4],%f1
1723         st      %f1,[%o0+0x4]
1724         st      %f0,[%o0+0x8]
1725         .end

1727         .inline __r_dim,2
1728         st      %g0,[%sp+0x48]
1729         ld      [%sp+0x48],%f4
1730         ld      [%o0],%f0
1731         ld      [%o1],%f2
1732         fcmps   %fcc0,%f0,%f2
1733         fmovsule %fcc0,%f4,%f2
1734         fsubs   %f0,%f2,%f0
1735         fmovsule %fcc0,%f4,%f0
1736         .end

1738         .inline __d_dim,2
1739         stx     %g0,[%sp+0x48]
1740         ldd     [%sp+0x48],%f4
1741         ld      [%o0],%f0
1742         ld      [%o0+4],%f1
1743         ld      [%o1],%f2
1744         ld      [%o1+4],%f3
1745         fcmpd   %fcc0,%f0,%f2
1746         fmovdule %fcc0,%f4,%f2
1747         fsubd   %f0,%f2,%f0
1748         fmovdule %fcc0,%f4,%f0
1749         .end

1751         .inline __r_imag,1
1752         ld      [%o0+4],%f0
1753         .end

1755         .inline __d_imag,1
1756         ld      [%o0+8],%f0
1757         ld      [%o0+0xc],%f1
1758         .end

1760         .inline __f95_signf,2
1761         ld      [%o0],%f0
1762         ld      [%o1],%o1
1763         fabss   %f0,%f0
1764         fnegs   %f0,%f1
1765         sra     %o1,0,%o1
1766         fmovrslz %o1,%f1,%f0
1767         .end

1769         .inline __f95_sign,2
1770         ld      [%o0],%f0
1771         ld      [%o0+4],%f1
1772         ld      [%o1],%o1
1773         fabsd   %f0,%f0
1774         fnegd   %f0,%f2
1775         sra     %o1,0,%o1
```



```
1776         fmovrdlz %o1,%f2,%f0
1777         .end

1779         .inline __r_sign,2
1780         ld      [%o0],%f0
1781         ld      [%o1],%o1
1782         fabss   %f0,%f0
1783         fnegs   %f0,%f1
1784         sub     %o1,1,%o0
1785         and     %o1,%o0,%o1
1786         sra     %o1,0,%o1
1787         fmovrslz %o1,%f1,%f0
1788         .end

1790         .inline __d_sign,2
1791         ld      [%o0],%f0
1792         ld      [%o0+4],%f1
1793         ld      [%o1],%o0
1794         sllx    %o0,32,%o0
1795         ld      [%o1+4],%o1
1796         or      %o1,%o0,%o1
1797         fabsd   %f0,%f0
1798         fnegd   %f0,%f2
1799         sub     %o1,1,%o0
1800         and     %o1,%o0,%o1
1801         fmovrdlz %o1,%f2,%f0
1802         .end

1804         .inline __Fz_mult,3
1805         ld      [%o1],%f0
1806         ld      [%o1+0x4],%f1
1807         ld      [%o2],%f4
1808         ld      [%o2+0x4],%f5
1809         fmuld   %f0,%f4,%f8     ! f8 = r1*r2
1810         ld      [%o1+0x8],%f2
1811         ld      [%o1+0xc],%f3
1812         ld      [%o2+0x8],%f6
1813         ld      [%o2+0xc],%f7
1814         fmuld   %f2,%f6,%f10    ! f10= i1*i2
1815         fsubd   %f8,%f10,%f12   ! f12= r1*r2-i1*i2
1816         st      %f12,[%o0]
1817         st      %f13,[%o0+4]
1818         fmuld   %f0,%f6,%f14    ! f14= r1*i2
1819         fmuld   %f2,%f4,%f16    ! f16= r2*i1
1820         faddd   %f14,%f16,%f2   ! f2 = r1*i2+r2*i1
1821         st      %f2,[%o0+8]
1822         st      %f3,[%o0+12]
1823         .end
1824 !- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
1825 ! void
1826 ! __Fc_minus(c, a, b)
1827 ! complex *c, *a, *b;
1828 ! {

1830         .inline __Fc_minus,3
1831 !    30         c->real = a->real - b->real
1832            ld      [%o1],%f0
1833            ld      [%o2],%f1
1834            fsubs   %f0,%f1,%f2
1835 !    31         c->imag = a->imag - b->imag
1836            ld      [%o1+4],%f3
1837            ld      [%o2+4],%f4
1838            fsubs   %f3,%f4,%f5
1839            st      %f2,[%o0]
1840            st      %f5,[%o0+4]
1841         .end
```

```
1842  }
1843 !- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
1844 ! void
1845 ! __Fc_add(c, a, b)
1846 ! complex *c, *a, *b;
1847 ! {

1849         .inline __Fc_add,3
1850 !    39         c->real = a->real + b->real
1851         ld      [%o1],%f0
1852         ld      [%o2],%f1
1853         fadds   %f0,%f1,%f2
1854 !    40         c->imag = a->imag + b->imag
1855         ld      [%o1+4],%f3
1856         ld      [%o2+4],%f4
1857         fadds   %f3,%f4,%f5
1858         st      %f2,[%o0]
1859         st      %f5,[%o0+4]
1860         .end
1861 ! }
1862 !- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
1863 ! void
1864 ! __Fc_neg(c, a)
1865 ! complex *c, *a;
1866 ! {

1868         .inline __Fc_neg,2
1869 !    48         c->real = - a->real
1870         ld      [%o1],%f0
1871         fnegs   %f0,%f1
1872 !    49         c->imag = - a->imag
1873         ld      [%o1+4],%f2
1874         fnegs   %f2,%f3
1875         st      %f1,[%o0]
1876         st      %f3,[%o0+4]
1877         .end
1878 ! }
1879 !- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
1880 ! void
1881 ! __Ff_conv_c(c, x)
1882 ! complex *c;
1883 ! FLOATPARAMETER x;
1884 ! {

1886         .inline __Ff_conv_c,2
1887 !    59         c->real = x
1888         st      %o1,[%o0]
1889 !    60         c->imag = 0.0
1890         st      %g0,[%o0+4]
1891         .end
1892 ! }
1893 !- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
1894 ! FLOATFUNCTIONTYPE
1895 ! __Fc_conv_f(c)
1896 ! complex *c;
1897 ! {

1899         .inline __Fc_conv_f,1
1900 !    69         RETURNFLOAT(c->real)
1901         ld      [%o0],%f0
1902         .end
1903 ! }
1904 !- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
1905 ! int
1906 ! __Fc_conv_i(c)
1907 ! complex *c;
```



```
1908 ! {

1910         .inline __Fc_conv_i,1
1911 !    78         return (int)c->real
1912         ld      [%o0],%f0
1913         fstoi   %f0,%f1
1914         st      %f1,[%sp+68]
1915         ld      [%sp+68],%o0
1916         .end
1917 ! }
1918 !- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
1919 ! void
1920 ! __Fi_conv_c(c, i)
1921 ! complex *c;
1922 ! int i;
1923 ! {

1925         .inline __Fi_conv_c,2
1926 !    88         c->real = (float)i
1927         st      %o1,[%sp+68]
1928         ld      [%sp+68],%f0
1929         fitos   %f0,%f1
1930         st      %f1,[%o0]
1931 !    89         c->imag = 0.0
1932         st      %g0,[%o0+4]
1933         .end
1934 ! }
1935 !- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
1936 ! double
1937 ! __Fc_conv_d(c)
1938 ! complex *c;
1939 ! {

1941         .inline __Fc_conv_d,1
1942 !    98         return (double)c->real
1943         ld      [%o0],%f2
1944         fstod   %f2,%f0
1945         .end
1946 ! }
1947 !- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
1948 ! void
1949 ! __Fd_conv_c(c, x)
1950 ! complex *c;
1951 ! double x;
1952 ! {

1954         .inline __Fd_conv_c,2
1955         st      %o1,[%sp+72]
1956         st      %o2,[%sp+76]
1957 !   109         c->real = (float)(x)
1958         ldd     [%sp+72],%f0
1959         fdtos   %f0,%f1
1960         st      %f1,[%o0]
1961 !   110         c->imag = 0.0
1962         st      %g0,[%o0+4]
1963         .end
1964 ! }
1965 !- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
1966 ! void
1967 ! __Fc_conv_z(result, c)
1968 ! dcomplex *result;
1969 ! complex *c;
1970 ! {

1972         .inline __Fc_conv_z,2
1973 !   120         result->dreal = (double)c->real
```

```
1974         ld      [%o1],%f0
1975         fstod   %f0,%f2
1976         st      %f2,[%o0]
1977         st      %f3,[%o0+4]
1978 !   121         result->dimag = (double)c->imag
1979         ld      [%o1+4],%f3
1980         fstod   %f3,%f4
1981         st      %f4,[%o0+8]
1982         st      %f5,[%o0+12]
1983         .end
1984 ! }
1985 !- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
1986 ! int
1987 ! __Fc_eq(x, y)
1988 ! complex *x, *y;
1989 ! {

1991         .inline __Fc_eq,2
1992 !       return  (x->real == y->real) && (x->imag == y->imag);
1993         ld      [%o0],%f0
1994         ld      [%o1],%f2
1995         mov     %o0,%o2
1996         fcmps   %f0,%f2
1997         mov     0,%o0
1998         fbne    1f
1999         nop
2000         ld      [%o2+4],%f0
2001         ld      [%o1+4],%f2
2002         fcmps   %f0,%f2
2003         nop
2004         fbne    1f
2005         nop
2006         mov     1,%o0
2007 1:
2008         .end
2009 ! }
2010 !- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
2011 ! int
2012 ! __Fc_ne(x, y)
2013 ! complex *x, *y;
2014 ! {

2016         .inline __Fc_ne,2
2017 !       return  (x->real != y->real) || (x->imag != y->imag);
2018         ld      [%o0],%f0
2019         ld      [%o1],%f2
2020         mov     %o0,%o2
2021         fcmps   %f0,%f2
2022         mov     1,%o0
2023         fbne    1f
2024         nop
2025         ld      [%o2+4],%f0
2026         ld      [%o1+4],%f2
2027         fcmps   %f0,%f2
2028         nop
2029         fbne    1f
2030         nop
2031         mov     0,%o0
2032 1:
2033         .end
2034 ! }
```

```
**********************************************************
    6471 Sun May  4 03:07:08 2014
new/usr/src/lib/libm/sparcv9/src/libm_inlines.h
**********************************************************
```

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

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

/*
 * Copyright 2011, Richard Lowe.
 */

/* Functions in this file are duplicated in locallibm.il.  Keep them in sync */
/* Functions in this file are duplicated in libm.m4.  Keep them in sync */

#ifndef _LIBM_INLINES_H
#define _LIBM_INLINES_H

#ifdef __GNUC__

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

#ifdef __cplusplus
extern "C" {
#endif

extern __inline__ enum fp_class_type
fp_classf(float f)
{
        enum fp_class_type ret;
        int fint;               /* scratch for f as int */
        uint64_t tmp;
#endif /* ! codereview */

        __asm__ __volatile__(
            "fabss  %3,%3\n\t"
            "st     %3,%1\n\t"
            "fabss  %2,%2\n\t"
            "st     %2,%1\n\t"
            "ld     %1,%0\n\t"
            "orcc   %%g0,%0,%%g0\n\t"
            "be,pn  %%icc,2f\n\t"
            "nop\n\t"
```



```
            "1:\n\t"
            "sethi  %%hi(0x7f800000),%2\n\t"
            "andcc  %0,%2,%%g0\n\t"
            "sethi  %%hi(0x7f800000),%%o1\n\t"
            "andcc  %0,%%o1,%%g0\n\t"
            "bne,pt %%icc,1f\n\t"
            "nop\n\t"
            "or     %%g0,1,%0\n\t"
            "ba     2f\n\t"     /* subnormal */
            "ba     2f\n\t"
            "nop\n\t"
            "1:\n\t"
            "subcc  %0,%2,%%g0\n\t"
            "subcc  %0,%%o1,%%g0\n\t"
            "bge,pn %%icc,1f\n\t"
            "nop\n\t"
            "or     %%g0,2,%0\n\t"
            "ba     2f\n\t"     /* normal */
            "ba     2f\n\t"
            "nop\n\t"
            "1:\n\t"
            "bg,pn  %%icc,1f\n\t"
            "nop\n\t"
            "or     %%g0,3,%0\n\t"
            "ba     2f\n\t"     /* infinity */
            "ba     2f\n\t"
            "nop\n\t"
            "1:\n\t"
            "sethi  %%hi(0x00400000),%2\n\t"
            "andcc  %0,%2,%%g0\n\t"
            "sethi  %%hi(0x00400000),%%o1\n\t"
            "andcc  %0,%%o1,%%g0\n\t"
            "or     %%g0,4,%0\n\t"
            "bne,pt %%icc,2f\n\t" /* quiet NaN */
            "bne,pt %%icc,2f\n\t"
            "nop\n\t"
            "or     %%g0,5,%0\n\t" /* signalling NaN */
            "or     %%g0,5,%0\n\t"
            "2:\n\t"
            : "=r" (ret), "=m" (fint), "=r" (tmp)
            : "=r" (ret), "=m" (fint)
            : "f" (f)
            : "cc");
            : "o1");

        return (ret);
}

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

        __asm__ __volatile__(
            "fabsd  %3,%3\n\t"
            "std    %3,%1\n\t"
            "fabsd  %2,%2\n\t"
            "std    %2,%1\n\t"
            "ldx    %1,%0\n\t"
            "orcc   %%g0,%0,%%g0\n\t"
            "be,pn  %%xcc,2f\n\t"
            "nop\n\t"
            "sethi  %%hi(0x7ff00000),%2\n\t"
```

```
112             "sllx   %2,32,%2\n\t"
113             "andcc  %0,%2,%%g0\n\t"
103             "sethi  %%hi(0x7ff00000),%%o1\n\t"
104             "sllx   %%o1,32,%%o1\n\t"
105             "andcc  %0,%%o1,%%g0\n\t"
114             "bne,pt %%xcc,1f\n\t"
115             "nop\n\t"
116             "or     %%g0,1,%0\n\t"
117             "ba     2f\n\t"
118             "nop\n\t"
119             "1:\n\t"
120             "subcc  %0,%2,%%g0\n\t"
112             "subcc  %0,%%o1,%%g0\n\t"
121             "bge,pn %%xcc,1f\n\t"
122             "nop\n\t"
123             "or     %%g0,2,%0\n\t"
124             "ba     2f\n\t"
125             "nop\n\t"
126             "1:\n\t"
127             "andncc %0,%2,%0\n\t"
119             "andncc %0,%%o1,%0\n\t"
128             "bne,pn %%xcc,1f\n\t"
129             "nop\n\t"
130             "or     %%g0,3,%0\n\t"
131             "ba     2f\n\t"
132             "nop\n\t"
133             "1:\n\t"
134             "sethi  %%hi(0x00080000),%2\n\t"
135             "sllx   %2,32,%2\n\t"
136             "andcc  %0,%2,%%g0\n\t"
126             "sethi  %%hi(0x00080000),%%o1\n\t"
127             "sllx   %%o1,32,%%o1\n\t"
128             "andcc  %0,%%o1,%%g0\n\t"
137             "or     %%g0,4,%0\n\t"
138             "bne,pt %%xcc,2f\n\t"
139             "nop\n\t"
140             "or     %%g0,5,%0\n\t"
141             "2:\n\t"
142             : "=r" (ret), "=m" (dint), "=r" (tmp)
134             : "=r" (ret), "=m" (dint)
143             : "e" (d)
144             : "cc");
136             : "o1");

146         return (ret);
147 }

149 extern __inline__ float
150 __inline_sqrtf(float f)
151 {
152         float ret;

154         __asm__ __volatile__("fsqrts %1,%0\n\t" : "=f" (ret) : "f" (f));
146         __asm__ __volatile__("fsqrts %0,%0\n\t" : "=f" (ret) : "f" (f));
155         return (ret);
156 }

158 extern __inline__ double
159 __inline_sqrt(double d)
160 {
161         double ret;

163         __asm__ __volatile__("fsqrtd %1,%0\n\t" : "=f" (ret) : "f" (d));
155         __asm__ __volatile__("fsqrtd %0,%0\n\t" : "=f" (ret) : "0" (d));
164         return (ret);
165 }
```



```
167 extern __inline__ int
168 __swapEX(int i)
169 {
170         int ret;
171         uint32_t fsr;
172         uint64_t tmp1, tmp2;
173 #endif /* ! codereview */

175         __asm__ __volatile__(
176             "and  %4,0x1f,%2\n\t"
177             "sll  %2,5,%2\n\t"  /* shift input to aexc bit location */
164             "and  %0,0x1f,%%o1\n\t"
165             "sll  %%o1,5,%%o1\n\t"    /* input to aexc bit location */
178             ".volatile\n\t"
179             "st   %%fsr,%1\n\t"
180             "ld   %1,%0\n\t"    /* %0 = fsr */
181             "andn %0,0x3e0,%3\n\t"
182             "or   %2,%3,%2\n\t" /* %2 = new fsr */
183             "st   %2,%1\n\t"
184             "ld   %1,%%fsr\n\t"
167             "st   %%fsr,%2\n\t"
168             "ld   %2,%0\n\t"          /* = fsr */
169             "andn %0,0x3e0,%%o2\n\t"
170             "or   %%o1,%%o2,%%o1\n\t" /* o1 = new fsr */
171             "st   %%o1,%2\n\t"
172             "ld   %2,%%fsr\n\t"
185             "srl  %0,5,%0\n\t"
186             "and  %0,0x1f,%0\n\t"
187             ".nonvolatile\n\t"
188             : "=r" (ret), "=m" (fsr), "=r" (tmp1), "=r" (tmp2)
189             : "r" (i)
190             : "cc");
176             : "=r" (ret)
177             : "0" (i), "m" (fsr)
178             : "o1", "o2");

192         return (ret);
193 }
```

*_____unchanged_portion_omitted_*

```
206 extern __inline__ enum fp_direction_type
207 __swapRD(enum fp_direction_type d)
208 {
209         enum fp_direction_type ret;
210         uint32_t fsr;
211         uint64_t tmp1, tmp2, tmp3;
212 #endif /* ! codereview */

214         __asm__ __volatile__(
215             "and   %5,0x3,%0\n\t"
216             "sll   %0,30,%2\n\t"        /* shift input to RD bit location */
199             "and   %0,0x3,%0\n\t"
200             "sll   %0,30,%%o1\n\t"            /* input to RD bit location */
217             ".volatile\n\t"
218             "st    %%fsr,%1\n\t"
219             "ld    %1,%0\n\t"           /* %0 = fsr */
220             /* mask of rounding direction bits */
221             "sethi %%hi(0xc0000000),%4\n\t"
222             "andn  %0,%4,%3\n\t"
223             "or    %2,%3,%2\n\t"        /* %2 = new fsr */
224             "st    %2,%1\n\t"
225             "ld    %1,%%fsr\n\t"
202             "st    %%fsr,%2\n\t"
203             "ld    %2,%0\n\t"                 /* o0 = fsr */
204             "sethi %%hi(0xc0000000),%%o4\n\t" /* mask of rounding direction bits
```

```
            "andn  %0,%%o4,%%o2\n\t"
            "or    %%o1,%%o2,%%o1\n\t"        /* o1 = new fsr */
            "st    %%o1,%2\n\t"
            "ld    %2,%%fsr\n\t"
            "srl   %0,30,%0\n\t"
            "and   %0,0x3,%0\n\t"
            ".nonvolatile\n\t"
            : "=r" (ret), "=m" (fsr), "=r" (tmp1), "=r" (tmp2), "=r" (tmp3)
            : "r" (d)
            : "cc");
            : "=r" (ret)
            : "0" (d), "m" (fsr)
            : "o1", "o2", "o4");

        return (ret);
}

extern __inline__ int
__swapTE(int i)
{
        int ret;
        uint32_t fsr;
        uint64_t tmp1, tmp2, tmp3;
#endif /* ! codereview */

        __asm__ __volatile__(
            "and   %5,0x1f,%0\n\t"
            "sll   %0,23,%2\n\t"        /* shift input to TEM bit location */
            "and   %0,0x1f,%0\n\t"
            "sll   %0,23,%%o1\n\t"            /* input to TEM bit location */
            ".volatile\n\t"
            "st    %%fsr,%1\n\t"
            "ld    %1,%0\n\t"           /* %0 = fsr */
            /* mask of TEM (Trap Enable Mode bits) */
            "sethi %%hi(0x0f800000),%4\n\t"
            "andn  %0,%4,%3\n\t"
            "or    %2,%3,%2\n\t"        /* %2 = new fsr */
            "st    %2,%1\n\t"
            "ld    %1,%%fsr\n\t"
            "st    %%fsr,%2\n\t"
            "ld    %2,%0\n\t"                /* o0 = fsr */
            "sethi %%hi(0x0f800000),%%o4\n\t" /* mask of TEM (Trap Enable Mode b
            "andn  %0,%%o4,%%o2\n\t"
            "or    %%o1,%%o2,%%o1\n\t"        /* o1 = new fsr */
            "st    %%o1,%2\n\t"
            "ld    %2,%%fsr\n\t"
            "srl   %0,23,%0\n\t"
            "and   %0,0x1f,%0\n\t"
            ".nonvolatile\n\t"
            : "=r" (ret), "=m" (fsr), "=r" (tmp1), "=r" (tmp2), "=r" (tmp3)
            : "r" (i)
            : "cc");
            : "=r" (ret)
            : "0" (i), "m" (fsr)
            : "o1", "o2", "o4");

        return (ret);
}


extern __inline__ double
sqrt(double d)
{
        return (__inline_sqrt(d));
    double ret;
```



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

extern __inline__ float
sqrtf(float f)
{
        return (__inline_sqrtf(f));
    float ret;

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

extern __inline__ double
fabs(double d)
{
        double ret;

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

extern __inline__ float
fabsf(float f)
{
        float ret;

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

```
**********************************************************
    5808 Sun May  4 03:07:10 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-unused-variable

176 #endif /* ! codereview */
177 LINTFLAGS       += $(LINTERROFF)
178 LINTFLAGS64     += $(LINTERROFF)
179 LINTFLAGS64     += -errchk=longptr64

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

184 ASDEF           += -DLIBMVEC_SO_BUILD

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

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



```
195 CPPFLAGS        += -DLIBMVEC_SO_BUILD

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

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

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

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

288 .KEEP_STATE:

290 all:    $(LIBS)

292 lint:   lintcheck

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

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

```
**********************************************************
    5808 Sun May  4 03:07:11 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-unused-variable

176 #endif /* ! codereview */
177 LINTFLAGS       += $(LINTERROFF)
178 LINTFLAGS64     += $(LINTERROFF)
179 LINTFLAGS64     += -errchk=longptr64

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

184 ASDEF           += -DLIBMVEC_SO_BUILD

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

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



```
195 CPPFLAGS        += -DLIBMVEC_SO_BUILD

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

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

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

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

288 .KEEP_STATE:

290 all:    $(LIBS)

292 lint:   lintcheck

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

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

```
**********************************************************
   11255 Sun May  4 03:07:13 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  4 03:07:14 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  4 03:07:16 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  4 03:07:18 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  4 03:07:21 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  4 03:07:23 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  4 03:07:25 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  4 03:07:26 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  4 03:07:29 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 = 0.63661977236758134307553505,  /* 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]= { -1.66666666666629669805215138920301589656 e-0001,
                                -4.99999999999931701464060878888294524481 e-0001
        poly2[2]= {  8.33333332390951295683993455280336376663 e-0003,
                                 4.16666666394861917535640593963708222319 e-0002
        poly3[2]= { -1.98412623799797669279155177823009840396 0e-0004,
                                -1.38888855265614286783275668773685168146 2e-0003
        poly4[2]= {  2.75340362485427723764998762284833035111 0e-0006,
                                 2.47851942368146079661812828945453052475 9e-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  4 03:07:30 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  4 03:07:31 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_***

```
**********************************************************
    1868 Sun May  4 03:07:33 2014
new/usr/src/man/Makefile
**********************************************************
#
# This file and its contents are supplied under the terms of the
# Common Development and Distribution License ("CDDL"), version 1.0.
# You may only use this file in accordance with the terms of version
# 1.0 of the CDDL.
#
# A full copy of the text of the CDDL should have accompanied this
# source.  A copy of the CDDL is also available via the Internet
# at http://www.illumos.org/license/CDDL.
#

#
# Copyright 2011, Richard Lowe
# Copyright (c) 2012, Igor Kozhukhov <ikozhukhov@gmail.com>
#endif /* ! codereview */
# Copyright 2013 Nexenta Systems, Inc.  All rights reserved.
#

SUBDIRS=        man1            \
                man1b           \
                man1c           \
                man1has         \
                man1m           \
                man2            \
                man3            \
                man3bsm         \
                man3c           \
                man3c_db        \
                man3cfgadm      \
                man3commputil   \
                man3contract    \
                man3cpc         \
                man3curses      \
                man3dat         \
                man3devid       \
                man3devinfo     \
                man3dlpi        \
                man3dns_sd      \
                man3elf         \
                man3exacct      \
                man3ext         \
                man3fcoe        \
                man3fstyp       \
                man3gen         \
                man3gss         \
                man3head        \
                man3iscsit      \
                man3kstat       \
                man3kvm         \
                man3ldap        \
                man3lgrp        \
                man3lib         \
                man3m           \
#endif /* ! codereview */
                man3mail        \
                man3malloc      \
                man3mp          \
                man3mpapi       \
                man3mvec        \
#endif /* ! codereview */
                man3nsl         \
                man3nvpair      \
```



```
                man3pam         \
                man3papi        \
                man3perl        \
                man3picl        \
                man3picltree    \
                man3pool        \
                man3proc        \
                man3project     \
                man3resolv      \
                man3rpc         \
                man3rsm         \
                man3sasl        \
                man3scf         \
                man3sec         \
                man3secdb       \
                man3sip         \
                man3slp         \
                man3socket      \
                man3stmf        \
                man3sysevent    \
                man3tecla       \
                man3tnf         \
                man3tsol        \
                man3uuid        \
                man3volmgt      \
                man3xcurses     \
                man3xnet        \
                man4            \
                man5            \
                man7            \
                man7d           \
                man7fs          \
                man7i           \
                man7ipp         \
                man7m           \
                man7p           \
                man9            \
                man9e           \
                man9f           \
                man9p           \
                man9s

.PARALLEL: $(SUBDIRS)

all             := TARGET = all
clean           := TARGET = clean
clobber         := TARGET = clobber
install         := TARGET = install

all clean clobber install: $(SUBDIRS)

$(SUBDIRS):     FRC
        @cd $@; pwd; $(MAKE) $(TARGET)

FRC:
```

```
**********************************************************
    1146 Sun May  4 03:07:35 2014
new/usr/src/pkg/manifests/SUNWlibm.mf
**********************************************************
```

```
#
# 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.
#

# was renamed to system/library/math/header-math
# both obsolete now
set name=pkg.fmri value=pkg:/SUNWlibm@0.5.11,5.11-0.132
set name=pkg.description \
    value="Math & Microtasking Library Headers & Lint Files"
# license license=SUNWlibm.copyright
# license license=SUNWlibmr.copyright
set name=pkg.renamed value=true
# set name=pkg.renamed value=true
set name=pkg.summary value="Math & Microtasking Library Headers & Lint Files"
set name=pkg.description value="Math Library Headers & Lint Files"
set name=pkg.obsolete value=true
set name=pkg.summary value="Math Library Headers & Lint Files"
set name=info.classification \
    value=org.opensolaris.category.2008:System/Libraries
# set name=org.opensolaris.consolidation value=sunpro
#endif /* ! codereview */
set name=variant.arch value=$(ARCH)
set name=variant.opensolaris.zone value=global value=nonglobal
depend fmri=pkg:/system/library/math/header-math@$(PKGVERS) type=require
#endif /* ! codereview */
```

```
**********************************************************
    1098 Sun May  4 03:07:37 2014
new/usr/src/pkg/manifests/SUNWlibms.mf
**********************************************************
```

```
   1 #
   2 # This file and its contents are supplied under the terms of the
   3 # Common Development and Distribution License ("CDDL"), version 1.0.
   4 # You may only use this file in accordance with the terms of version
   5 # 1.0 of the CDDL.
   6 #
   7 # A full copy of the text of the CDDL should have accompanied this
   8 # source.  A copy of the CDDL is also available via the Internet at
   9 # http://www.illumos.org/license/CDDL.
  10 #

  12 #
  13 # Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
  14 #

  16 set name=pkg.fmri value=pkg:/SUNWlibms@0.5.11,5.11-0.132
  17 set name=pkg.description value="Math & Microtasking Libraries"
  17 set name=pkg.description value="Math Libraries"
  18 # license license=SUNWlibms.copyright
  19 # license license=SUNWlibmsr.copyright
  20 set name=pkg.renamed value=true
  21 # set name=pkg.renamed value=true
  22 set name=pkg.summary value="Math & Microtasking Libraries"
  23 set name=description value="Math & Microtasking Libraries"
  22 set name=pkg.summary value="Math Libraries"
  23 set name=description value="Math Libraries"
  24 set name=info.classification \
  25     value=org.opensolaris.category.2008:System/Libraries
  26 set name=variant.arch value=$(ARCH)
  27 set name=variant.opensolaris.zone value=global value=nonglobal
  28 depend fmri=pkg:/system/library/math@$(PKGVERS) type=require
```

```
**********************************************************
   17025 Sun May  4 03:07:39 2014
new/usr/src/tools/aw/aw.c
**********************************************************
______unchanged_portion_omitted_

 480 int
 481 main(int argc, char *argv[])
 482 {
 483         struct aelist *cpp = NULL;
 484         struct aelist *m4 = NULL;
 485         struct aelist *as = newael();
 486         char **asargv;
 487         char *outfile = NULL;
 488         char *srcfile = NULL;
 489         const char *dir, *cmd;
 490         static char as_pgm[MAXPATHLEN];
 491         static char as64_pgm[MAXPATHLEN];
 492         static char m4_pgm[MAXPATHLEN];
 493         static char m4_cmdefs[MAXPATHLEN];
 494         static char cpp_pgm[MAXPATHLEN];
 495         int as64 = 0;
 496         int code;

 498         if ((progname = strrchr(argv[0], '/')) == NULL)
 499                 progname = argv[0];
 500         else
 501                 progname++;

 503         /*
 504          * Helpful when debugging, or when changing tool versions..
 505          */
 506         if ((cmd = getenv("AW_AS")) != NULL)
 507                 strlcpy(as_pgm, cmd, sizeof (as_pgm));
 508         else {
 509                 if ((dir = getenv("AW_AS_DIR")) == NULL)
 510                         dir = DEFAULT_AS_DIR;   /* /usr/sfw/bin */
 511                 (void) snprintf(as_pgm, sizeof (as_pgm), "%s/gas", dir);
 512         }

 514         if ((cmd = getenv("AW_AS64")) != NULL)
 515                 strlcpy(as64_pgm, cmd, sizeof (as64_pgm));
 516         else {
 517                 if ((dir = getenv("AW_AS64_DIR")) == NULL)
 518                         dir = DEFAULT_AS64_DIR; /* /usr/sfw/bin */
 519                 (void) snprintf(as64_pgm, sizeof (as_pgm), "%s/gas", dir);
 520         }

 522         if ((cmd = getenv("AW_M4")) != NULL)
 523                 strlcpy(m4_pgm, cmd, sizeof (m4_pgm));
 524         else {
 525                 if ((dir = getenv("AW_M4_DIR")) == NULL)
 526                         dir = DEFAULT_M4_DIR;   /* /usr/ccs/bin */
 527                 (void) snprintf(m4_pgm, sizeof (m4_pgm), "%s/m4", dir);
 528         }

 530         if ((cmd = getenv("AW_M4LIB")) != NULL)
 531                 strlcpy(m4_cmdefs, cmd, sizeof (m4_cmdefs));
 532         else {
 533                 if ((dir = getenv("AW_M4LIB_DIR")) == NULL)
 534                         dir = DEFAULT_M4LIB_DIR;        /* /usr/ccs/lib */
 535                 (void) snprintf(m4_cmdefs, sizeof (m4_cmdefs),
 536                     "%s/cm4defs", dir);
 537         }

 539         if ((cmd = getenv("AW_CPP")) != NULL)
```



```
 540                 strlcpy(cpp_pgm, cmd, sizeof (cpp_pgm));
 541         else {
 542                 if ((dir = getenv("AW_CPP_DIR")) == NULL)
 543                         dir = DEFAULT_CPP_DIR;  /* /usr/ccs/lib */
 544                 (void) snprintf(cpp_pgm, sizeof (cpp_pgm), "%s/cpp", dir);
 545         }

 547         newae(as, as_pgm);
 548         newae(as, "--warn");
 549         newae(as, "--fatal-warnings");
 550         newae(as, "--traditional-format");

 552         /*
 553          * Walk the argument list, translating as we go ..
 554          */
 555         while (--argc > 0) {
 556                 char *arg;
 557                 int arglen;

 559                 arg = *++argv;
 560                 arglen = strlen(arg);

 562                 if (*arg != '-') {
 563                         char *filename;

 565                         /*
 566                          * filenames ending in '.s' are taken to be
 567                          * assembler files, and provide the default
 568                          * basename of the output file.
 569                          *
 570                          * other files are passed through to the
 571                          * preprocessor, if present, or to gas if not.
 572                          */
 573                         filename = arg;
 574                         if ((arglen > 2) &&
 575                             ((strcmp(arg + arglen - 2, ".s") == 0) ||
 576                             (strcmp(arg + arglen - 2, ".S") == 0))) {
 574                         if (arglen > 2 &&
 575                             (strcmp(arg + arglen - 2, ".s") == 0) ||
 576                             (strcmp(arg + arglen - 2, ".S") == 0)) {
 577                                 /*
 578                                  * Though 'as' allows multiple assembler
 579                                  * files to be processed in one invocation
 580                                  * of the assembler, ON only processes one
 581                                  * file at a time, which makes things a lot
 582                                  * simpler!
 583                                  */
 584                                 if (srcfile == NULL)
 585                                         srcfile = arg;
 586                                 else
 587                                         return (usage(
 588                                             "one assembler file at a time"));

 590                                 /*
 591                                  * If we haven't seen a -o option yet,
 592                                  * default the output to the basename
 593                                  * of the input, substituting a .o on the end
 594                                  */
 595                                 if (outfile == NULL) {
 596                                         char *argcopy;

 598                                         argcopy = strdup(arg);
 599                                         argcopy[arglen - 1] = 'o';

 601                                         if ((outfile = strrchr(
 602                                             argcopy, '/')) == NULL)
```

```
603                                             outfile = argcopy;
604                                     else
605                                             outfile++;
606                             }
607                     }
608                     if (cpp)
609                             newae(cpp, filename);
610                     else if (m4)
611                             newae(m4, filename);
612                     else
613                             newae(as, filename);
614                     continue;
615             } else
616                     arglen--;

618             switch (arg[1]) {
619             case 'K':
620                     /*
621                      * -K pic
622                      * -K PIC
623                      */
624                     if (arglen == 1) {
625                             if ((arg = *++argv) == NULL || *arg == '\0')
626                                     return (usage("malformed -K"));
627                             argc--;
628                     } else {
629                             arg += 2;
630                     }
631                     if (strcmp(arg, "PIC") != 0 && strcmp(arg, "pic") != 0)
632                             return (usage("malformed -K"));
633                     break;          /* just ignore -Kpic for gcc */
634             case 'Q':
635                     if (strcmp(arg, "-Qn") == 0)
636                             break;
637                     /*FALLTHROUGH*/
638             case 'b':
639             case 's':
640             case 'T':
641                     /*
642                      * -b   Extra symbol table for source browser ..
643                      *      not relevant to gas, thus should error.
644                      * -s   Put stabs in .stabs section not stabs.excl
645                      *      not clear if there's an equivalent
646                      * -T   4.x migration option
647                      */
648             default:
649                     return (error(arg));
650             case 'x':
651                     /*
652                      * Accept -xarch special case to invoke alternate
653                      * assemblers or assembler flags for different
654                      * architectures.
655                      */
656                     if (strcmp(arg, "-xarch=amd64") == 0 ||
657                         strcmp(arg, "-xarch=generic64") == 0) {
658                             as64++;
659                             fixae_arg(as->ael_head, as64_pgm);
660                             break;
661                     }
662                     /*
663                      * XX64: Is this useful to gas?
664                      */
665                     if (strcmp(arg, "-xmodel=kernel") == 0)
666                             break;

668                     /*
```



```
669                      * -xF  Generates performance analysis data
670                      *      no equivalent
671                      */
672                     return (error(arg));
673             case 'V':
674                     newae(as, arg);
675                     break;
676             case '#':
677                     verbose++;
678                     break;
679             case 'L':
680                     newae(as, "--keep-locals");
681                     break;
682             case 'n':
683                     newae(as, "--no-warn");
684                     break;
685             case 'o':
686                     if (arglen != 1)
687                             return (usage("bad -o flag"));
688                     if ((arg = *++argv) == NULL || *arg == '\0')
689                             return (usage("bad -o flag"));
690                     outfile = arg;
691                     argc--;
692                     arglen = strlen(arg + 1);
693                     break;
694             case 'm':
695                     if (cpp)
696                             return (usage("-m conflicts with -P"));
697                     if (m4 == NULL) {
698                             m4 = newael();
699                             newae(m4, m4_pgm);
700                             newae(m4, m4_cmdefs);
701                     }
702                     break;
703             case 'P':
704                     if (m4)
705                             return (usage("-P conflicts with -m"));
706                     if (cpp == NULL) {
707                             cpp = newael();
708                             newae(cpp, cpp_pgm);
709                             newae(cpp, "-D__GNUC_AS__");
710                     }
711                     break;
712             case 'D':
713             case 'U':
714                     if (cpp)
715                             newae(cpp, arg);
716                     else if (m4)
717                             newae(m4, arg);
718                     else
719                             newae(as, arg);
720                     break;
721             case 'I':
722                     if (cpp)
723                             newae(cpp, arg);
724                     else
725                             newae(as, arg);
726                     break;
727             case '-':       /* a gas-specific option */
728                     newae(as, arg);
729                     break;
730             }
731     }

733 #if defined(__i386)
734     if (as64)
```

```
735                 newae(as, "--64");
736         else
737                 newae(as, "--32");
738 #endif

740         if (srcfile == NULL)
741                 return (usage("no source file(s) specified"));
742         if (outfile == NULL)
743                 outfile = "a.out";
744         newae(as, "-o");
745         newae(as, outfile);

747         asargv = aeltoargv(as);
748         if (cpp) {
749 #if defined(__sparc)
750                 newae(cpp, "-Dsparc");
751                 newae(cpp, "-D__sparc");
752                 if (as64)
753                         newae(cpp, "-D__sparcv9");
754                 else
755                         newae(cpp, "-D__sparcv8");
756 #elif defined(__i386) || defined(__x86)
757                 if (as64) {
758                         newae(cpp, "-D__x86_64");
759                         newae(cpp, "-D__amd64");
760                 } else {
761                         newae(cpp, "-Di386");
762                         newae(cpp, "-D__i386");
763                 }
764 #else
765 #error  "need isa-dependent defines"
766 #endif
767                 code = pipeline(aeltoargv(cpp), asargv);
768         } else if (m4)
769                 code = pipeline(aeltoargv(m4), asargv);
770         else {
771                 /*
772                  * XXX  should arrange to fork/exec so that we
773                  *      can unlink the output file if errors are
774                  *      detected..
775                  */
776                 (void) execvp(asargv[0], asargv);
777                 perror("execvp");
778                 (void) fprintf(stderr, "%s: couldn't run %s\n",
779                     progname, asargv[0]);
780                 code = 7;
781         }
782         if (code != 0)
783                 (void) unlink(outfile);
784         return (code);
785 }
```

*unchanged_portion_omitted_*

```
**********************************************************
   45780 Sun May  4 03:07:41 2014
new/usr/src/tools/cw/cw.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, Richard Lowe.
*/
/*
 * Copyright 2010 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

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

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

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



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

```
 * -xinline=[<a>,...,<a>]  Attempt inlining of specified user routines,
 *              <a>={%auto,func,no%func}
 * -xlibmieee   Force IEEE 754 return values for math routines in
 *              exceptional cases
 * -xlibmil     Inline selected libm math routines for optimization
 * -xlic_lib=sunperf    Link in the Sun supplied performance libraries
 * -xlicinfo    Show license server information
 * -xM          Generate makefile dependencies
 * -xM1         Generate makefile dependencies, but exclude /usr/include
 * -xmaxopt=[off,1,2,3,4,5] maximum optimization level allowed on #pragma opt
 * -xnolib      Do not link with default system libraries
 * -xnolibmil   Cancel -xlibmil on command line
 * -xO<n>       Generate optimized code (n={1|2|3|4|5})
 * -xP          Print prototypes for function definitions
 * -xpentium    Generate code for the pentium processor
 * -xpg         Compile for profiling with gprof
 * -xprofile=<p> Collect data for a profile or use a profile to optimize
 *              <p>={{collect,use}[:<path>],tcov}
 * -xregs=<r>   Control register allocation
 * -xs          Allow debugging without object (.o) files
 * -xsb         Compile for use with the WorkShop source browser
 * -xsbfast     Generate only WorkShop source browser info, no compilation
 * -xsfpconst   Represent unsuffixed floating point constants as single
 *              precision
 * -xspace      Do not do optimizations that increase code size
 * -xstrconst   Place string literals into read-only data segment
 * -xtarget=<t> Specify target system for optimization
 * -xtemp=<dir> Set directory for temporary files to <dir>
 * -xtime       Report the execution time for each compilation phase
 * -xtransition Emit warnings for differences between K&R C and ANSI C
 * -xtrigraphs[=<yes|no>] Enable|disable trigraph translation
 * -xunroll=n   Enable unrolling loops n times where possible
 * -Y<c>,<dir>  Specify <dir> for location of component <c> (a,l,m,p,0,h,i,u)
 * -YA,<dir>    Change default directory searched for components
 * -YI,<dir>    Change default directory searched for include files
 * -YP,<dir>    Change default directory for finding libraries files
 * -YS,<dir>    Change default directory for startup object files
 */

/*
 * Translation table:
 */
/*
 * -#                           -v
 * -###                         error
 * -A<name[(tokens)]>           pass-thru
 * -B<[static|dynamic]>         pass-thru (syntax error for anything else)
 * -C                           pass-thru
 * -c                           pass-thru
 * -cg92                        -m32 -mcpu=v8 -mtune=supersparc (SPARC only)
 * -D<name[=token]>             pass-thru
 * -dy or -dn                   -Wl,-dy or -Wl,-dn
 * -E                           pass-thru
 * -erroff=E_EMPTY_TRANSLATION_UNIT ignore
 * -errtags=%all                -Wall
 * -errwarn=%all                -Werror else -Wno-error
 * -fast                        error
 * -fd                          error
 * -features=zla                ignore
 * -flags                       --help
 * -fnonstd                     error
 * -fns[=<yes|no>]              error
 * -fprecision=<p>              error
 * -fround=<r>                  error
 * -fsimple[=<n>]               error
 * -fsingle[=<n>]               error
```



```
 * -ftrap=<t>                   error
 * -fstore                      error
 * -G                           pass-thru
 * -g                           pass-thru
 * -H                           pass-thru
 * -h <name>                    pass-thru
 * -I<dir>                      pass-thru
 * -i                           pass-thru
 * -keeptmp                     -save-temps
 * -KPIC                        -fPIC
 * -Kpic                        -fpic
 * -L<dir>                      pass-thru
 * -l<name>                     pass-thru
 * -mc                          error
 * -mr                          error
 * -mr,"string"                 error
 * -mt                          -D_REENTRANT
 * -native                      error
 * -nofstore                    error
 * -nolib                       -nodefaultlibs
 * -noqueue                     ignore
 * -norunpath                   ignore
 * -O                           -O1 (Check the man page to be certain)
 * -o <outputfile>              pass-thru
 * -P                           -E -o filename.i (or error)
 * -PIC                         -fPIC (C++ only)
 * -p                           pass-thru
 * -pic                         -fpic (C++ only)
 * -Q[y|n]                      error
 * -qp                          -p
 * -R<dir[:dir]>                pass-thru
 * -S                           pass-thru
 * -s                           -Wl,-s
 * -t                           -Wl,-t
 * -U<name>                     pass-thru
 * -V                           --version
 * -v                           -Wall
 * -Wa,<arg>                    pass-thru
 * -Wp,<arg>                    pass-thru except -xc99=<a>
 * -Wl,<arg>                    pass-thru
 * -W{m,0,2,h,i,u>              error/ignore
 * -Wu,-xmodel=kernel           -ffreestanding -mcmodel=kernel -mno-red-zone
 * -xmodel=kernel               -ffreestanding -mcmodel=kernel -mno-red-zone
 * -Wu,-save_args               -msave-args
 * -w                           pass-thru
 * -Xa                          -std=iso9899:199409 or -ansi
 * -Xc                          -ansi -pedantic
 * -Xt                          error
 * -Xs                          -traditional -std=c89
 * -x386                        -march=i386 (x86 only)
 * -x486                        -march=i486 (x86 only)
 * -xarch=<a>                   table
 * -xbuiltin[=<b>]              -fbuiltin (-fno-builtin otherwise)
 * -xCC                         ignore
 * -xchar_byte_order=<o>        error
 * -xchip=<c>                   table
 * -xcode=<c>                   table
 * -xdebugformat=<format>       ignore (always use dwarf-2 for gcc)
 * -xcrossfile[=<n>]            ignore
 * -xe                          error
 * -xF                          error
 * -xhelp=<f>                   error
 * -xildoff                     ignore
 * -xildon                      ignore
 * -xinline                     ignore
 * -xlibmieee                   error
```

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

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

 306 #define CW_F_CXX        0x01
 307 #define CW_F_SHADOW     0x02
 308 #define CW_F_EXEC       0x04
 309 #define CW_F_ECHO       0x08
 310 #define CW_F_XLATE      0x10
 311 #define CW_F_PROG       0x20

 313 typedef enum cw_compiler {
 314         CW_C_CC = 0,
 315         CW_C_GCC
 316 } cw_compiler_t;
```

*_____unchanged_portion_omitted_*