```
   MODULE FMVALS_PARALLEL
      PRIVATE AINT, CEILING, DIGITS, EPSILON, HUGE, LOG, MAX, MIN, SQRT


!     Version 1.4                        David M. Smith                              9-01-2021

!  This is the thread-safe version of FM.  It can be used with a program that uses coarrays,
!  which is the Fortran-standard way of parallel programming.  It should also be ok to use
!  with other, non-standard, ways of parallel programming (openMP, MPI, etc.).

!  Because of restrictions imposed by the need to be thread-safe, this version of FM has
!  several limitations compared to the regular version of FM.

!  1.  Global variables defined in modules are not thread-safe if they can be changed
!      while the program runs.  This means the user sets the FM precision level by
!      defining variable FM_SIGNIFICANT_DIGITS here in module FMVALS_PARALLEL, then
!      compiles this file and links it to their program.  Calling FM_SET to set the
!      precision is not available.

!  2.  TYPE(IM) integer multi-precision numbers have varying numbers of digits that are
!      indirectly limited by FM_SIGNIFICANT_DIGITS.  Make sure that all TYPE(IM) values,
!      intermediate as well as final results, have fewer than 20*FM_SIGNIFICANT_DIGITS
!      decimal digits.

!  3.  It is less common, but if a user program wants different values for other FM
!      options like rounding mode, screen width for FM output, etc., they must be
!      initialized here in this module and not changed while the program runs.

!  4.  Similarly, FM precision level cannot be changed by the user's program while
!      it runs.

!  5.  The FM_RANDOM_NUMBER random number generator cannot be used, since it depends
!      on a global saved state to get the next number.

!  6.  Because multi-precision variables are no longer stored in a global module
!      database, the FM_DEALLOCATE function is not needed and has been removed.
!      Similarly, the FM_(ENTER or EXIT)_USER_(FUNCTION or ROUTINE) calls are not
!      needed and those routines have been removed.

!  7.  Saved values like pi, e, euler gamma, etc., are global variables in the standard
!      version of FM.  They have been removed from this thread-safe version, so they are
!      re-computed each time they are needed.  That makes some functions like trig and
!      log/exponential functions slightly slower.

!  8.  The global allocatable database from version 1.3 has been replaced with local fixed-size
!      arrays for the multiple precision numbers.  A few routines like fm_fprime and zm_fprime
!      that relied on raising precision a lot to overcome unstable formulas may now return unknown.
!      Up to 4th or 5th derivatives should usually be ok, but higher derivatives may fail.


      INTEGER, PARAMETER :: FM_SIGNIFICANT_DIGITS = 50

!  Default settings for other FM options:
!  MBASE is set to a power of 10, 10**7 for machines with 64-bit double precision.
!  The trace option is set off.
!  The mode for angles in trig functions is set to radians.
!  The rounding mode is set to symmetric rounding (to nearest).
```

```
!  Warning error message level is set to 1.
!  Screen width for output is set to 100 columns.
!  The exponent character for FM output is set to 'M'.
!  Debug error checking is set off.
!  KW, the unit number for all FM output, is set to 6.



      REAL (KIND(1.0D0)), PARAMETER :: M_TWO = 2
      DOUBLE PRECISION, PARAMETER :: DP_TWO = 2
      INTEGER, PARAMETER :: I_TWO = 2
      REAL, PARAMETER :: R_TWO = 2

!             KW is the unit number for standard output from the FM package.
!                This includes trace output and error messages.

      INTEGER, PARAMETER :: KW = 6

      REAL (KIND(1.0D0)), PARAMETER :: MAX_REPRESENTABLE_M_VAR =  &
                           ( (M_TWO ** (MIN(DIGITS(M_TWO),DIGITS(DP_TWO))-1)) - 1 ) * 2 + 1

      REAL (KIND(1.0D0)), PARAMETER :: MAXINT = MAX_REPRESENTABLE_M_VAR

      INTEGER, PARAMETER :: INTMAX = HUGE(I_TWO)

      DOUBLE PRECISION, PARAMETER :: DPMAX = HUGE(DP_TWO)/5

      REAL, PARAMETER :: SPMAX = HUGE(R_TWO)/5

      REAL (KIND(1.0D0)), PARAMETER :: MAX_BASE = AINT(SQRT(MAX_REPRESENTABLE_M_VAR + 1.0D-9))

      REAL (KIND(1.0D0)), PARAMETER :: MXBASE = MAX_BASE

      REAL (KIND(1.0D0)), PARAMETER :: M_TEN = 10

      REAL (KIND(1.0D0)), PARAMETER :: MAX_EXPONENT = AINT( MIN(                                  &
                                       MAX(HUGE(INTMAX) / LOG(MAX_BASE+1.0D-9) , 117496405.0D0),  &
                                       MAX_REPRESENTABLE_M_VAR / 20.0D0) )

      INTEGER, PARAMETER :: LHASH1 =   0
      INTEGER, PARAMETER :: LHASH2 = 256

      INTEGER, PARAMETER :: LJSUMS = 100

      INTEGER, PARAMETER :: LMBERN = 600

      INTEGER, PARAMETER :: NDIG_USER = CEILING( (FM_SIGNIFICANT_DIGITS+2) /  &
                                              AINT(LOG(MAX_BASE/4.0D0)/LOG(10.0D0)) ) + 1

      INTEGER, PARAMETER :: NDIG_MAX = 25*NDIG_USER

      INTEGER, PARAMETER :: LMBUFF = MAX(25*FM_SIGNIFICANT_DIGITS+50,550)
      INTEGER, PARAMETER :: LMBUFZ = MAX(50*FM_SIGNIFICANT_DIGITS+50,1000)

      TYPE MULTI
         REAL (KIND(1.0D0)) :: MP(NDIG_MAX+2)
      END TYPE
```

```
TYPE WORK_AREA
   REAL (KIND(1.0D0)) :: MP(2*NDIG_MAX+20)
END TYPE

TYPE FM
   TYPE(MULTI) :: MFM
END TYPE

TYPE IM
   TYPE(MULTI) :: MIM
END TYPE

TYPE ZM
   TYPE(MULTI) :: MZM(2)
END TYPE

TYPE FM_SETTINGS
   INTEGER :: NDIG = NDIG_USER
   INTEGER :: KFLAG = 0
   INTEGER :: NTRACE = 0
   INTEGER :: LVLTRC = 1
   INTEGER :: NCALL = 0
   REAL (KIND(1.0D0)) :: MBASE = M_TEN ** AINT(LOG(MAX_BASE/4.0D0) / LOG(10.0D0))
   INTEGER :: RAISE_NDIG = 0
   CHARACTER(9) :: NAMEST(0:50)
   INTEGER :: NDIGPI = 0
   REAL (KIND(1.0D0)) :: MBSPI = 0
   INTEGER :: NDIGE = 0
   REAL (KIND(1.0D0)) :: MBSE = 0
   INTEGER :: NDIGLB = 0
   REAL (KIND(1.0D0)) :: MBSLB = 0
   INTEGER :: NDIGLI = 0
   REAL (KIND(1.0D0)) :: MBSLI = 0
   REAL (KIND(1.0D0)) :: MXEXP = AINT( MAX_EXPONENT / 2.01D0 + 0.5D0 )
   REAL (KIND(1.0D0)) :: MXEXP2 = MAX_EXPONENT
   REAL (KIND(1.0D0)) :: MEXPUN = AINT( -MAX_EXPONENT * 1.01D0 )
   REAL (KIND(1.0D0)) :: MEXPOV = AINT( MAX_EXPONENT * 1.01D0 )
   REAL (KIND(1.0D0)) :: MUNKNO = AINT( MAX_EXPONENT * 1.0201D0 )
   REAL :: RUNKNO = -1.01*(HUGE(R_TWO)/3.0)
   INTEGER :: IUNKNO = -HUGE(I_TWO)/18
   INTEGER :: JFORM1 = 1
   INTEGER :: JFORM2 = FM_SIGNIFICANT_DIGITS
   INTEGER :: KRAD = 1
   INTEGER :: KWARN = 1
   INTEGER :: KROUND = 1
   INTEGER :: KSWIDE = 100
   INTEGER :: KESWCH = 1
   CHARACTER :: CMCHAR = 'M'
   INTEGER :: KDEBUG = 0
   INTEGER :: KROUND_RETRY = 0
   INTEGER :: KSUB = 0
   INTEGER :: KSQR = 0
   INTEGER :: KREM = 1
   INTEGER :: JRSIGN = 0
   INTEGER :: LHASH = 0
   INTEGER :: KHASHT(LHASH1:LHASH2)
   INTEGER :: KHASHV(LHASH1:LHASH2)
   DOUBLE PRECISION :: DPEPS = EPSILON(DP_TWO)
```

```
        REAL (KIND(1.0D0)) :: MBLOGS = 0
        REAL :: ALOGMB = 1.611810E+1
        REAL :: ALOGM2 = 2.325350E+1
        REAL :: ALOGMX = 3.673680E+1
        REAL :: ALOGMT = 7.0E0
        INTEGER :: NGRD21 = 1
        INTEGER :: NGRD52 = 2
        INTEGER :: NGRD22 = 2
        REAL (KIND(1.0D0)) :: MEXPAB = AINT(MAX_EXPONENT / 5.0D0)
        DOUBLE PRECISION :: DLOGMB = 1.611809565095832D+1
        DOUBLE PRECISION :: DLOGTN = 2.302585092994046D+0
        DOUBLE PRECISION :: DLOGTW = 6.931471805599453D-1
        DOUBLE PRECISION :: DPPI   = 3.141592653589793D+0
        DOUBLE PRECISION :: DLOGTP = 1.837877066409345D+0
        DOUBLE PRECISION :: DLOGPI = 1.144729885849400D+0
        DOUBLE PRECISION :: DLOGEB = 2.236222824932432D+0
        REAL (KIND(1.0D0)) :: MBASEL = 0
        REAL (KIND(1.0D0)) :: MBASEN = 0
        INTEGER :: NDIGL = 0
        INTEGER :: NDIGN = 0
        INTEGER :: NGUARL = 0
        INTEGER :: N21
        INTEGER :: NGRDN
        INTEGER :: JFORMZ = 1
        INTEGER :: JPRNTZ = 1
        REAL (KIND(1.0D0)) :: MBS2PI = 0
        INTEGER :: NDG2PI = 0
    END TYPE

  END MODULE FMVALS_PARALLEL


    SUBROUTINE FMSET(NPREC,QX)

!  The user cannot change FM settings in the thread-safe version.

    USE FMVALS_PARALLEL
    IMPLICIT NONE

    INTEGER :: NPREC
    TYPE(FM_SETTINGS) :: QX

    WRITE (*,"(//A)")  ' The user cannot change FM settings in the thread-safe version.'
    IF (NPREC >= -31 .OR. QX%MBASE > -31) THEN
        WRITE (*,"(/A//)") ' See the comments at the top of file FM_parallel.f95'
    ENDIF
    STOP

    END SUBROUTINE FMSET

    SUBROUTINE FMABS(MA,MB,QX)

!  MB = ABS(MA)

    USE FMVALS_PARALLEL
    IMPLICIT NONE

    TYPE(MULTI) :: MA,MB
```

```
      INTENT (IN) :: MA
      INTENT (INOUT) :: MB
      TYPE(FM_SETTINGS) :: QX

      INTEGER :: KWRNSV


      QX%NCALL = QX%NCALL + 1
      QX%NAMEST(QX%NCALL) = 'FMABS'
      IF (QX%NTRACE /= 0) CALL FMNTR(2,MA,MA,1,1,QX)

      QX%KFLAG = 0
      KWRNSV = QX%KWARN
      QX%KWARN = 0
      CALL FMEQ(MA,MB,QX)
      MB%MP(1) = 1
      QX%KWARN = KWRNSV
      IF (QX%NTRACE /= 0) CALL FMNTR(1,MB,MB,1,1,QX)
      QX%NCALL = QX%NCALL - 1
      RETURN
      END SUBROUTINE FMABS

      SUBROUTINE FMACOS(MA,MB,QX)

!  MB = ACOS(MA)

      USE FMVALS_PARALLEL
      IMPLICIT NONE

      TYPE(MULTI) :: MA,MB
      REAL (KIND(1.0D0)) :: MAS,MXSAVE
      DOUBLE PRECISION :: ERR
      INTEGER :: J,K,KL,KOVUN,KR_RETRY,KRESLT,NDSAVE
      INTENT (IN) :: MA
      INTENT (INOUT) :: MB
      TYPE(MULTI) :: MXY(6)
      TYPE(FM_SETTINGS) :: QX


      IF (QX%MBLOGS /= QX%MBASE) CALL FMCONS(QX)

!             Rounding for special cases in non-standard rounding modes (KROUND = -1, 0, or 2).

      IF (QX%KROUND /= 1 .AND. MA%MP(2) < -QX%NDIG .AND. QX%KRAD == 0) THEN
          IF (QX%NTRACE /= 0) THEN
              QX%NCALL = QX%NCALL + 1
              QX%NAMEST(QX%NCALL) = 'FMACOS'
              CALL FMNTR(2,MA,MA,1,1,QX)
              QX%NCALL = QX%NCALL - 1
          ENDIF
          J = QX%NTRACE
          QX%NTRACE = 0
          K = QX%KWARN
          QX%KWARN = 0
          CALL FMI2M(180,MXY(1),QX)
          CALL FMPI(MXY(2),QX)
          CALL FMDIV(MXY(1),MXY(2),MXY(3),QX)
          IF (MA%MP(2) > QX%MEXPUN) THEN
```

```
            CALL FMMPY(MXY(3),MA,MXY(2),QX)
        ELSE
            CALL FMEQ(MA,MXY(2),QX)
        ENDIF
        CALL FMI2M(90,MXY(1),QX)
        CALL FMSUB(MXY(1),MXY(2),MB,QX)
        IF (QX%KFLAG > 0) QX%KFLAG = 0
        QX%NTRACE = J
        QX%KWARN = K
        IF (MB%MP(2) == QX%MUNKNO .AND. MA%MP(2) /= QX%MUNKNO) THEN
            QX%NCALL = QX%NCALL + 1
            QX%NAMEST(QX%NCALL) = 'FMACOS'
            QX%KFLAG = -4
            CALL FMWARN(QX)
            QX%NCALL = QX%NCALL - 1
        ELSE IF (ABS(MB%MP(2)) == QX%MEXPOV .AND. ABS(MA%MP(2)) < QX%MEXPOV) THEN
            QX%NCALL = QX%NCALL + 1
            QX%NAMEST(QX%NCALL) = 'FMACOS'
            IF (MB%MP(2) == QX%MEXPOV) QX%KFLAG = -5
            IF (MB%MP(2) == QX%MEXPUN) QX%KFLAG = -6
            CALL FMWARN(QX)
            QX%NCALL = QX%NCALL - 1
        ENDIF
        IF (QX%NTRACE /= 0) THEN
            QX%NCALL = QX%NCALL + 1
            QX%NAMEST(QX%NCALL) = 'FMACOS'
            CALL FMNTR(1,MB,MB,1,1,QX)
            QX%NCALL = QX%NCALL - 1
        ENDIF
        RETURN
    ENDIF
    IF (ABS(MA%MP(2)) > QX%MEXPAB .OR. MA%MP(2) > 0 .OR. MA%MP(3) == 0) THEN
        CALL FMENTR('FMACOS   ',MA,MA,1,1,MB,KRESLT,NDSAVE,MXSAVE,KOVUN,QX)
        IF (KRESLT /= 0) THEN
            RETURN
        ENDIF
    ELSE
        QX%NCALL = QX%NCALL + 1
        QX%NAMEST(QX%NCALL) = 'FMACOS'
        IF (QX%NTRACE /= 0) CALL FMNTR(2,MA,MA,1,1,QX)
        KOVUN = 0
        IF (MA%MP(2) == QX%MEXPOV .OR. MA%MP(2) == QX%MEXPUN) KOVUN = 1
        NDSAVE = QX%NDIG
        IF (QX%NCALL == 1) THEN
            K = MAX(QX%NGRD52-1,2)
            QX%NDIG = MAX(QX%NDIG+K,2)
        ENDIF
        MXSAVE = QX%MXEXP
        QX%MXEXP = QX%MXEXP2
    ENDIF
    KR_RETRY = 0

110 IF (KR_RETRY >= 1) THEN
        IF (QX%NCALL == 1) QX%NDIG = MAX(QX%NDIG,2*NDSAVE+10)
    ENDIF
    MAS = MA%MP(1)
    CALL FMEQU(MA,MXY(6),NDSAVE,QX%NDIG,QX)
    IF (QX%KROUND /= 1 .AND. QX%KRAD /= 1) THEN
```

```
          CALL FMST2M('0.5',MXY(1),QX)
          CALL FMSUB(MXY(6),MXY(1),MXY(2),QX)
          IF (MXY(2)%MP(3) == 0) THEN
              CALL FMST2M('60',MXY(6),QX)
              GO TO 120
          ENDIF
          CALL FMADD(MXY(6),MXY(1),MXY(2),QX)
          IF (MXY(2)%MP(3) == 0) THEN
              CALL FMST2M('120',MXY(6),QX)
              GO TO 120
          ENDIF
      ENDIF

!             Use ACOS(X) = ATAN(SQRT(1-X*X)/X)

      MXY(6)%MP(1) = 1
      CALL FMI2M(1,MXY(4),QX)
      CALL FMSUB(MXY(4),MXY(6),MXY(2),QX)
      CALL FMADD(MXY(4),MXY(6),MXY(3),QX)
      CALL FMMPY_R2(MXY(2),MXY(3),QX)
      CALL FMSQRT_R1(MXY(3),QX)
      CALL FMDIV(MXY(3),MXY(6),MXY(5),QX)
      CALL FMATAN(MXY(5),MXY(6),QX)

      IF (MAS < 0) THEN
          IF (QX%KRAD == 1) THEN
              CALL FMPI(MXY(4),QX)
          ELSE
              CALL FMI2M(180,MXY(4),QX)
          ENDIF
          CALL FMSUB_R2(MXY(4),MXY(6),QX)
      ENDIF

!             Round the result and return.

!             Try again with more guard digits if the current guard digits are too close to 1/2 ulp.

  120 IF (QX%NCALL >= 1) THEN
          KL = MIN(QX%NDIG-NDSAVE,INT(3*QX%DLOGTN/QX%DLOGMB + 1.5))
          ERR = 0
          DO J = KL, 1, -1
             ERR = (ERR + MXY(6)%MP(J+NDSAVE+2)) / QX%MBASE
          ENDDO
          IF ( (QX%KROUND == 1 .AND. ERR > 0.498 .AND. ERR < 0.502) .OR.  &
               (QX%KROUND /= 1 .AND. (ERR > 0.998 .OR. ERR < 0.002)) ) KR_RETRY = KR_RETRY + 1
      ENDIF
      IF (KR_RETRY == 1 .AND. QX%NDIG < 2*NDSAVE+10) THEN
          KR_RETRY = 2
          GO TO 110
      ENDIF
      CALL FMEXIT(MXY(6),MB,NDSAVE,MXSAVE,KOVUN,QX)
      RETURN
      END SUBROUTINE FMACOS

      SUBROUTINE FMACOSH(MA,MB,QX)

!  MB = ARCCOSH(MA)      Inverse hyperbolic cosine.
```

```
      USE FMVALS_PARALLEL
      IMPLICIT NONE

      TYPE(MULTI) :: MA,MB
      TYPE(MULTI) :: MLN2,MLN3,MLN5,MLN7
      REAL (KIND(1.0D0)) :: MXSAVE
      DOUBLE PRECISION :: ERR
      INTEGER :: IEXTRA,J,KL,KOVUN,KR_RETRY,KRESLT,NDSAVE
      INTENT (IN) :: MA
      INTENT (INOUT) :: MB
      TYPE(MULTI) :: MXY(4)
      TYPE(FM_SETTINGS) :: QX


      CALL FMENTR('FMACOSH  ',MA,MA,1,1,MB,KRESLT,NDSAVE,MXSAVE,KOVUN,QX)
      IF (KRESLT /= 0) THEN
          RETURN
      ENDIF
      KR_RETRY = 0

  110 IF (KR_RETRY >= 1) THEN
          IF (QX%NCALL == 1) QX%NDIG = MAX(QX%NDIG,2*NDSAVE+10)
      ENDIF
      CALL FMEQU(MA,MXY(1),NDSAVE,QX%NDIG,QX)

      IF (MXY(1)%MP(2) == QX%MEXPOV .OR. MXY(1)%MP(2) <= 0 .OR.  &
          MXY(1)%MP(1) < 0) THEN
          QX%KFLAG = -4
          CALL FMST2M('UNKNOWN',MXY(2),QX)
      ELSE IF (4.0*(MXY(1)%MP(2)-1) > QX%NDIG) THEN
          CALL FMMPYI(MXY(1),2,MXY(2),QX)
          IF (MXY(2)%MP(2) == QX%MEXPOV) THEN
              CALL FMLN(MXY(1),MXY(2),QX)
              CALL FMLNI(2,MXY(3),MLN2,MLN3,MLN5,MLN7,QX)
              CALL FMADD_R1(MXY(2),MXY(3),QX)
          ELSE
              CALL FMI2M(1,MXY(3),QX)
              CALL FMSQR(MXY(2),MXY(4),QX)
              CALL FMDIV_R2(MXY(3),MXY(4),QX)
              CALL FMLN(MXY(2),MXY(3),QX)
              CALL FMSUB(MXY(3),MXY(4),MXY(2),QX)
          ENDIF
      ELSE
          IEXTRA = MXY(1)%MP(2)
          IF (IEXTRA > 0) THEN
              CALL FMEQU_R1(MXY(1),QX%NDIG,QX%NDIG+IEXTRA,QX)
              QX%NDIG = QX%NDIG + IEXTRA
          ENDIF
          CALL FMI2M(1,MXY(2),QX)
          CALL FMSUB(MXY(1),MXY(2),MXY(3),QX)
          CALL FMADD(MXY(1),MXY(2),MXY(4),QX)
          CALL FMMPY_R1(MXY(3),MXY(4),QX)
          CALL FMSQRT_R1(MXY(3),QX)
          CALL FMDIV_R1(MXY(3),MXY(1),QX)
          CALL FMATANH(MXY(3),MXY(2),QX)
      ENDIF
      QX%NAMEST(QX%NCALL) = 'FMACOSH'
```

```
!             Try again with more guard digits if the current guard digits are too close to 1/2 ulp.

      IF (QX%NCALL >= 1) THEN
          KL = MIN(QX%NDIG-NDSAVE,INT(3*QX%DLOGTN/QX%DLOGMB + 1.5))
          ERR = 0
          DO J = KL, 1, -1
             ERR = (ERR + MXY(2)%MP(J+NDSAVE+2)) / QX%MBASE
          ENDDO
          IF ( (QX%KROUND == 1 .AND. ERR > 0.498 .AND. ERR < 0.502) .OR.  &
               (QX%KROUND /= 1 .AND. (ERR > 0.998 .OR. ERR < 0.002)) ) KR_RETRY = KR_RETRY + 1
      ENDIF
      IF (KR_RETRY == 1 .AND. QX%NDIG < 2*NDSAVE+10) THEN
          KR_RETRY = 2
          GO TO 110
      ENDIF

      CALL FMEXIT(MXY(2),MB,NDSAVE,MXSAVE,KOVUN,QX)
      RETURN
      END SUBROUTINE FMACOSH

      SUBROUTINE FMADD(MA,MB,MC,QX)

!  MC = MA + MB

!  This routine performs the trace printing for addition.  FMADD2 is used to do the arithmetic.

      USE FMVALS_PARALLEL
      IMPLICIT NONE

      TYPE(MULTI) :: MA,MB,MC
      INTENT (IN) :: MA,MB
      INTENT (INOUT) :: MC
      TYPE(FM_SETTINGS) :: QX

      QX%NCALL = QX%NCALL + 1
      IF (QX%NTRACE /= 0) THEN
          QX%NAMEST(QX%NCALL) = 'FMADD'
          CALL FMNTR(2,MA,MB,2,1,QX)

          CALL FMADD2(MA,MB,MC,QX)

          CALL FMNTR(1,MC,MC,1,1,QX)
      ELSE
          CALL FMADD2(MA,MB,MC,QX)
      ENDIF
      QX%NCALL = QX%NCALL - 1
      RETURN
      END SUBROUTINE FMADD

      SUBROUTINE FMADD2(MA,MB,MC,QX)

!  Internal addition routine.  MC = MA + MB

      USE FMVALS_PARALLEL
      IMPLICIT NONE

      TYPE(MULTI) :: MA,MB,MC
      TYPE(FM_SETTINGS) :: QX
```

```
      TYPE(WORK_AREA) :: MWA
      REAL (KIND(1.0D0)) :: MAS,MBS
      INTEGER :: J,JCOMP,JRSSAV,JSIGN,KRESLT,N1,NGUARD,NMWA
      INTENT (IN) :: MA,MB
      INTENT (INOUT) :: MC


      IF (QX%MBLOGS /= QX%MBASE) CALL FMCONS(QX)
      JRSSAV = QX%JRSIGN
      IF (ABS(MA%MP(2)) > QX%MEXPAB .OR. ABS(MB%MP(2)) > QX%MEXPAB .OR. QX%KDEBUG == 1) THEN
          IF (QX%KSUB == 1) THEN
              CALL FMARGS('FMSUB    ',2,MA,MB,KRESLT,QX)
          ELSE
              CALL FMARGS('FMADD    ',2,MA,MB,KRESLT,QX)
          ENDIF
          IF (KRESLT /= 0) THEN
              IF ((KRESLT /= 1 .AND. KRESLT /= 2) .OR. MA%MP(3) == 0 .OR.  &
                  MB%MP(3) == 0) THEN
                  QX%NCALL = QX%NCALL + 1
                  IF (QX%KSUB == 1) THEN
                      QX%NAMEST(QX%NCALL) = 'FMSUB'
                  ELSE
                      QX%NAMEST(QX%NCALL) = 'FMADD'
                  ENDIF
                  CALL FMRSLT(MA,MB,MC,KRESLT,QX)
                  QX%JRSIGN = JRSSAV
                  QX%NCALL = QX%NCALL - 1
                  RETURN
              ENDIF
          ENDIF
      ELSE
          IF (MA%MP(3) == 0) THEN
              CALL FMEQ(MB,MC,QX)
              QX%KFLAG = 1
              IF (QX%KSUB == 1) THEN
                  IF (MC%MP(2) /= QX%MUNKNO .AND. MC%MP(3) /= 0)  &
                      MC%MP(1) = -MC%MP(1)
                  QX%KFLAG = 0
              ENDIF
              QX%JRSIGN = JRSSAV
              RETURN
          ENDIF
          IF (MB%MP(3) == 0) THEN
              CALL FMEQ(MA,MC,QX)
              QX%KFLAG = 1
              QX%JRSIGN = JRSSAV
              RETURN
          ENDIF
      ENDIF
      QX%KFLAG = 0
      N1 = QX%NDIG + 1

!             NGUARD is the number of guard digits used.

  110 IF (QX%NCALL > 1) THEN
          NGUARD = QX%NGRD21
          IF (NGUARD > QX%NDIG) NGUARD = QX%NDIG
```

```
      ELSE
          NGUARD = QX%NGRD52
          IF (NGUARD > QX%NDIG) NGUARD = QX%NDIG
          IF (QX%KROUND_RETRY >= 1) THEN
              NGUARD = QX%NDIG
          ENDIF
      ENDIF
      NMWA = N1 + NGUARD

!             Save the signs of MA and MB and then work with positive numbers.
!             JSIGN is the sign of the result of MA + MB.

      JSIGN = 1
      MAS = MA%MP(1)
      MBS = MB%MP(1)
      IF (QX%KSUB == 1) MBS = -MBS

!             See which one is larger in absolute value.

      JCOMP = 2
      IF (MA%MP(2) > MB%MP(2)) THEN
          JCOMP = 1
      ELSE IF (MB%MP(2) > MA%MP(2)) THEN
          JCOMP = 3
      ELSE
          DO J = 2, N1
             IF (MA%MP(J+1) > MB%MP(J+1)) THEN
                 JCOMP = 1
                 EXIT
             ENDIF
             IF (MB%MP(J+1) > MA%MP(J+1)) THEN
                 JCOMP = 3
                 EXIT
             ENDIF
          ENDDO
      ENDIF

      IF (JCOMP < 3) THEN
          IF (MAS < 0) JSIGN = -1
          QX%JRSIGN = JSIGN
          IF (MAS*MBS > 0) THEN
              CALL FMADDP(MA,MB,MWA,NGUARD,NMWA,QX)
          ELSE
              CALL FMADDN(MA,MB,MWA,NGUARD,NMWA,QX)
          ENDIF
      ELSE
          IF (MBS < 0) JSIGN = -1
          QX%JRSIGN = JSIGN
          IF (MAS*MBS > 0) THEN
              CALL FMADDP(MB,MA,MWA,NGUARD,NMWA,QX)
          ELSE
              CALL FMADDN(MB,MA,MWA,NGUARD,NMWA,QX)
          ENDIF
      ENDIF

!             Try again with more guard digits if the current guard digits are too close to 1/2 ulp.

      IF (QX%KROUND_RETRY == 1 .AND. NGUARD < QX%NDIG) THEN
```

```
          QX%KROUND_RETRY = 2
          GO TO 110
       ENDIF

!             Transfer to MC and fix the sign of the result.

       CALL FMMOVE(MWA,MC,QX)
       MC%MP(1) = 1
       IF (JSIGN < 0 .AND. MC%MP(3) /= 0) MC%MP(1) = -1

       IF (QX%KFLAG < 0) THEN
          IF (QX%KSUB == 1) THEN
             QX%NAMEST(QX%NCALL) = 'FMSUB'
          ELSE
             QX%NAMEST(QX%NCALL) = 'FMADD'
          ENDIF
          CALL FMWARN(QX)
       ENDIF

       QX%JRSIGN = JRSSAV
       QX%KROUND_RETRY = 0
       RETURN
       END SUBROUTINE FMADD2

       SUBROUTINE FMADD_R1(MA,MB,QX)

!  MA = MA + MB

!  This routine performs the trace printing for addition.  FMADD2_R1 is used to do the arithmetic.

       USE FMVALS_PARALLEL
       IMPLICIT NONE

       TYPE(MULTI) :: MA,MB
       INTENT (INOUT) :: MA
       INTENT (IN) :: MB
       TYPE(FM_SETTINGS) :: QX

       QX%NCALL = QX%NCALL + 1
       IF (QX%NTRACE /= 0) THEN
          QX%NAMEST(QX%NCALL) = 'FMADD_R1'
          CALL FMNTR(2,MA,MB,2,1,QX)

          CALL FMADD2_R1(MA,MB,QX)

          QX%NAMEST(QX%NCALL) = 'FMADD_R1'
          CALL FMNTR(1,MA,MA,1,1,QX)
       ELSE
          CALL FMADD2_R1(MA,MB,QX)
       ENDIF
       QX%NCALL = QX%NCALL - 1
       RETURN
       END SUBROUTINE FMADD_R1

       SUBROUTINE FMADD2_R1(MA,MB,QX)

!  Internal addition routine.  MA = MA + MB
```

```
USE FMVALS_PARALLEL
IMPLICIT NONE

TYPE(MULTI) :: MA,MB
TYPE(WORK_AREA) :: MWA
TYPE(FM_SETTINGS) :: QX

REAL (KIND(1.0D0)) :: MAS,MBS
INTEGER :: J,JCOMP,JRSSAV,JSIGN,KRESLT,N1,NGUARD,NMWA
INTENT (INOUT) :: MA
INTENT (IN) :: MB
TYPE(MULTI) :: MXY(2)


IF (QX%MBLOGS /= QX%MBASE) CALL FMCONS(QX)
JRSSAV = QX%JRSIGN
IF (ABS(MA%MP(2)) > QX%MEXPAB .OR. ABS(MB%MP(2)) > QX%MEXPAB .OR. QX%KDEBUG == 1) THEN
    IF (QX%KSUB == 1) THEN
        CALL FMARGS('FMSUB    ',2,MA,MB,KRESLT,QX)
    ELSE
        CALL FMARGS('FMADD    ',2,MA,MB,KRESLT,QX)
    ENDIF
    IF (KRESLT /= 0) THEN
        IF ((KRESLT /= 1 .AND. KRESLT /= 2) .OR. MA%MP(3) == 0 .OR.  &
            MB%MP(3) == 0) THEN
            QX%NCALL = QX%NCALL + 1
            IF (QX%KSUB == 1) THEN
                QX%NAMEST(QX%NCALL) = 'FMSUB_R1'
            ELSE
                QX%NAMEST(QX%NCALL) = 'FMADD_R1'
            ENDIF
            CALL FMRSLT(MA,MB,MXY(1),KRESLT,QX)
            CALL FMEQ(MXY(1),MA,QX)
            QX%JRSIGN = JRSSAV
            QX%NCALL = QX%NCALL - 1
            RETURN
        ENDIF
    ENDIF
ELSE
    IF (MA%MP(3) == 0) THEN
        CALL FMEQ(MB,MA,QX)
        QX%KFLAG = 1
        IF (QX%KSUB == 1) THEN
            IF (MA%MP(2) /= QX%MUNKNO .AND. MA%MP(3) /= 0)  &
                MA%MP(1) = -MA%MP(1)
            QX%KFLAG = 0
        ENDIF
        QX%JRSIGN = JRSSAV
        RETURN
    ENDIF
    IF (MB%MP(3) == 0) THEN
        QX%KFLAG = 1
        QX%JRSIGN = JRSSAV
        RETURN
    ENDIF
ENDIF
QX%KFLAG = 0
N1 = QX%NDIG + 1
```

```
!             NGUARD is the number of guard digits used.

  110 IF (QX%NCALL > 1) THEN
          NGUARD = QX%NGRD21
          IF (NGUARD > QX%NDIG) NGUARD = QX%NDIG
      ELSE
          NGUARD = QX%NGRD52
          IF (NGUARD > QX%NDIG) NGUARD = QX%NDIG
          IF (QX%KROUND_RETRY >= 1) THEN
              NGUARD = QX%NDIG
          ENDIF
      ENDIF
      NMWA = N1 + NGUARD

!             Save the signs of MA and MB and then work with positive numbers.
!             JSIGN is the sign of the result of MA + MB.

      JSIGN = 1
      MAS = MA%MP(1)
      MBS = MB%MP(1)
      IF (QX%KSUB == 1) MBS = -MBS

!             See which one is larger in absolute value.

      JCOMP = 2
      IF (MA%MP(2) > MB%MP(2)) THEN
          JCOMP = 1
      ELSE IF (MB%MP(2) > MA%MP(2)) THEN
          JCOMP = 3
      ELSE
          DO J = 2, N1
             IF (MA%MP(J+1) > MB%MP(J+1)) THEN
                 JCOMP = 1
                 EXIT
             ENDIF
             IF (MB%MP(J+1) > MA%MP(J+1)) THEN
                 JCOMP = 3
                 EXIT
             ENDIF
          ENDDO
      ENDIF

      IF (JCOMP < 3) THEN
          IF (MAS < 0) JSIGN = -1
          QX%JRSIGN = JSIGN
          IF (MAS*MBS > 0) THEN
              CALL FMADDP(MA,MB,MWA,NGUARD,NMWA,QX)
          ELSE
              CALL FMADDN(MA,MB,MWA,NGUARD,NMWA,QX)
          ENDIF
      ELSE
          IF (MBS < 0) JSIGN = -1
          QX%JRSIGN = JSIGN
          IF (MAS*MBS > 0) THEN
              CALL FMADDP(MB,MA,MWA,NGUARD,NMWA,QX)
          ELSE
              CALL FMADDN(MB,MA,MWA,NGUARD,NMWA,QX)
```

```
          ENDIF
        ENDIF

!               Try again with more guard digits if the current guard digits are too close to 1/2 ulp.

        IF (QX%KROUND_RETRY == 1 .AND. NGUARD < QX%NDIG) THEN
            QX%KROUND_RETRY = 2
            GO TO 110
        ENDIF

!               Transfer to MA and fix the sign of the result.

        CALL FMMOVE(MWA,MA,QX)
        MA%MP(1) = 1
        IF (JSIGN < 0 .AND. MA%MP(3) /= 0) MA%MP(1) = -1

        IF (QX%KFLAG < 0) THEN
            IF (QX%KSUB == 1) THEN
                QX%NAMEST(QX%NCALL) = 'FMSUB_R1'
            ELSE
                QX%NAMEST(QX%NCALL) = 'FMADD_R1'
            ENDIF
            CALL FMWARN(QX)
        ENDIF

        QX%JRSIGN = JRSSAV
        QX%KROUND_RETRY = 0
        RETURN
        END SUBROUTINE FMADD2_R1

        SUBROUTINE FMADD_R2(MA,MB,QX)

!  MB = MA + MB

!  This routine performs the trace printing for addition.  FMADD2_R2 is used to do the arithmetic.

        USE FMVALS_PARALLEL
        IMPLICIT NONE

        TYPE(MULTI) :: MA,MB
        INTENT (IN) :: MA
        INTENT (INOUT) :: MB
        TYPE(FM_SETTINGS) :: QX

        QX%NCALL = QX%NCALL + 1
        IF (QX%NTRACE /= 0) THEN
            QX%NAMEST(QX%NCALL) = 'FMADD_R2'
            CALL FMNTR(2,MA,MB,2,1,QX)

            CALL FMADD2_R2(MA,MB,QX)

            QX%NAMEST(QX%NCALL) = 'FMADD_R2'
            CALL FMNTR(1,MB,MB,1,1,QX)
        ELSE
            CALL FMADD2_R2(MA,MB,QX)
        ENDIF
        QX%NCALL = QX%NCALL - 1
        RETURN
```

```
      END SUBROUTINE FMADD_R2

      SUBROUTINE FMADD2_R2(MA,MB,QX)

!  Internal addition routine.  MB = MA + MB

      USE FMVALS_PARALLEL
      IMPLICIT NONE

      TYPE(MULTI) :: MA,MB
      TYPE(WORK_AREA) :: MWA
      TYPE(FM_SETTINGS) :: QX

      REAL (KIND(1.0D0)) :: MAS,MBS
      INTEGER :: J,JCOMP,JRSSAV,JSIGN,KRESLT,N1,NGUARD,NMWA
      INTENT (IN) :: MA
      INTENT (INOUT) :: MB
      TYPE(MULTI) :: MXY(2)


      IF (QX%MBLOGS /= QX%MBASE) CALL FMCONS(QX)
      JRSSAV = QX%JRSIGN
      IF (ABS(MA%MP(2)) > QX%MEXPAB .OR. ABS(MB%MP(2)) > QX%MEXPAB .OR. QX%KDEBUG == 1) THEN
          IF (QX%KSUB == 1) THEN
              CALL FMARGS('FMSUB    ',2,MA,MB,KRESLT,QX)
          ELSE
              CALL FMARGS('FMADD    ',2,MA,MB,KRESLT,QX)
          ENDIF
          IF (KRESLT /= 0) THEN
              IF ((KRESLT /= 1 .AND. KRESLT /= 2) .OR. MA%MP(3) == 0 .OR.  &
                  MB%MP(3) == 0) THEN
                  QX%NCALL = QX%NCALL + 1
                  IF (QX%KSUB == 1) THEN
                      QX%NAMEST(QX%NCALL) = 'FMSUB_R2'
                  ELSE
                      QX%NAMEST(QX%NCALL) = 'FMADD_R2'
                  ENDIF
                  CALL FMRSLT(MA,MB,MXY(1),KRESLT,QX)
                  CALL FMEQ(MXY(1),MB,QX)
                  QX%JRSIGN = JRSSAV
                  QX%NCALL = QX%NCALL - 1
                  RETURN
              ENDIF
          ENDIF
      ELSE
          IF (MA%MP(3) == 0) THEN
              QX%KFLAG = 1
              IF (QX%KSUB == 1) THEN
                  IF (MB%MP(2) /= QX%MUNKNO .AND. MB%MP(3) /= 0)  &
                      MB%MP(1) = -MB%MP(1)
                  QX%KFLAG = 0
              ENDIF
              QX%JRSIGN = JRSSAV
              RETURN
          ENDIF
          IF (MB%MP(3) == 0) THEN
              CALL FMEQ(MA,MB,QX)
              QX%KFLAG = 1
```

```
            QX%JRSIGN = JRSSAV
            RETURN
         ENDIF
      ENDIF
      QX%KFLAG = 0
      N1 = QX%NDIG + 1

!             NGUARD is the number of guard digits used.

  110 IF (QX%NCALL > 1) THEN
         NGUARD = QX%NGRD21
         IF (NGUARD > QX%NDIG) NGUARD = QX%NDIG
      ELSE
         NGUARD = QX%NGRD52
         IF (NGUARD > QX%NDIG) NGUARD = QX%NDIG
         IF (QX%KROUND_RETRY >= 1) THEN
            NGUARD = QX%NDIG
         ENDIF
      ENDIF
      NMWA = N1 + NGUARD

!             Save the signs of MA and MB and then work with positive numbers.
!             JSIGN is the sign of the result of MA + MB.

      JSIGN = 1
      MAS = MA%MP(1)
      MBS = MB%MP(1)
      IF (QX%KSUB == 1) MBS = -MBS

!             See which one is larger in absolute value.

      JCOMP = 2
      IF (MA%MP(2) > MB%MP(2)) THEN
         JCOMP = 1
      ELSE IF (MB%MP(2) > MA%MP(2)) THEN
         JCOMP = 3
      ELSE
         DO J = 2, N1
            IF (MA%MP(J+1) > MB%MP(J+1)) THEN
               JCOMP = 1
               EXIT
            ENDIF
            IF (MB%MP(J+1) > MA%MP(J+1)) THEN
               JCOMP = 3
               EXIT
            ENDIF
         ENDDO
      ENDIF

      IF (JCOMP < 3) THEN
         IF (MAS < 0) JSIGN = -1
         QX%JRSIGN = JSIGN
         IF (MAS*MBS > 0) THEN
            CALL FMADDP(MA,MB,MWA,NGUARD,NMWA,QX)
         ELSE
            CALL FMADDN(MA,MB,MWA,NGUARD,NMWA,QX)
         ENDIF
      ELSE
```

```
          IF (MBS < 0) JSIGN = -1
          QX%JRSIGN = JSIGN
          IF (MAS*MBS > 0) THEN
              CALL FMADDP(MB,MA,MWA,NGUARD,NMWA,QX)
          ELSE
              CALL FMADDN(MB,MA,MWA,NGUARD,NMWA,QX)
          ENDIF
      ENDIF

!             Try again with more guard digits if the current guard digits are too close to 1/2 ulp.

      IF (QX%KROUND_RETRY == 1 .AND. NGUARD < QX%NDIG) THEN
          QX%KROUND_RETRY = 2
          GO TO 110
      ENDIF

!             Transfer to MB and fix the sign of the result.

      CALL FMMOVE(MWA,MB,QX)
      MB%MP(1) = 1
      IF (JSIGN < 0 .AND. MB%MP(3) /= 0) MB%MP(1) = -1

      IF (QX%KFLAG < 0) THEN
          IF (QX%KSUB == 1) THEN
              QX%NAMEST(QX%NCALL) = 'FMSUB_R2'
          ELSE
              QX%NAMEST(QX%NCALL) = 'FMADD_R2'
          ENDIF
          CALL FMWARN(QX)
      ENDIF

      QX%JRSIGN = JRSSAV
      QX%KROUND_RETRY = 0
      RETURN
      END SUBROUTINE FMADD2_R2

      SUBROUTINE FMADDI(MA,IVAL,QX)

!  MA = MA + IVAL

!  Increment MA by one word integer IVAL.

!  This routine is faster than FMADD when IVAL is small enough so that it can be added to a single
!  word of MA without often causing a carry.  Otherwise FMI2M and FMADD are used.

      USE FMVALS_PARALLEL
      IMPLICIT NONE

      TYPE(MULTI) :: MA
      INTEGER :: IVAL
      REAL (KIND(1.0D0)) :: MAEXP,MKSUM
      INTEGER :: KPTMA
      INTENT (INOUT) :: MA
      INTENT (IN) :: IVAL
      TYPE(MULTI) :: MXY(2)
      TYPE(FM_SETTINGS) :: QX
```

```
      QX%NCALL = QX%NCALL + 1
      IF (QX%NTRACE /= 0) THEN
          QX%NAMEST(QX%NCALL) = 'FMADDI'
          CALL FMNTR(2,MA,MA,1,1,QX)
          CALL FMNTRI(2,IVAL,0,QX)
      ENDIF
      QX%KFLAG = 0

      MAEXP = MA%MP(2)
      IF (MAEXP <= 0 .OR. MAEXP > QX%NDIG) GO TO 110
      KPTMA = INT(MAEXP) + 1
      IF (MA%MP(1) < 0) THEN
          MKSUM = MA%MP(KPTMA+1) - IVAL
      ELSE
          MKSUM = MA%MP(KPTMA+1) + IVAL
      ENDIF

      IF (MKSUM >= QX%MBASE .OR. MKSUM < 0) GO TO 110
      IF (KPTMA == 2 .AND. MKSUM == 0) GO TO 110
      MA%MP(KPTMA+1) = MKSUM
      GO TO 120

  110 CALL FMI2M(IVAL,MXY(1),QX)
      CALL FMADD2_R1(MA,MXY(1),QX)

  120 IF (QX%NTRACE /= 0) THEN
          CALL FMNTR(1,MA,MA,1,1,QX)
      ENDIF
      QX%NCALL = QX%NCALL - 1
      RETURN
      END SUBROUTINE FMADDI

      SUBROUTINE FMADDN(MA,MB,MWA,NGUARD,NMWA,QX)

!  Internal addition routine.  MWA = MA - MB
!  The arguments are such that MA >= MB >= 0.

!  NGUARD is the number of guard digits being carried.
!  NMWA is the number of words in MWA that will be used.

      USE FMVALS_PARALLEL
      IMPLICIT NONE

      TYPE(MULTI) :: MA,MB
      INTEGER :: NGUARD,NMWA
      TYPE(WORK_AREA) :: MWA
      REAL (KIND(1.0D0)) :: MK,MR
      DOUBLE PRECISION :: ERR
      INTEGER :: J,K,KL,KP1,KP2,KPT,KSH,N1,N2,NK,NK1
      INTENT (IN) :: MA,MB
      TYPE(FM_SETTINGS) :: QX

      N1 = QX%NDIG + 1

!             Check for an insignificant operand.

      MK = MA%MP(2) - MB%MP(2)
      IF (MK >= QX%NDIG+2) THEN
```

```
          DO J = 1, N1
             MWA%MP(J+1) = MA%MP(J+1)
          ENDDO
          MWA%MP(N1+2) = 0
          IF (QX%KROUND == 0 .OR. (QX%KROUND == 2 .AND. QX%JRSIGN == -1) .OR.  &
             (QX%KROUND == -1 .AND. QX%JRSIGN == 1)) THEN
              MWA%MP(N1+1) = MWA%MP(N1+1) - 1
              GO TO 120
          ENDIF
          QX%KFLAG = 1
          RETURN
       ENDIF
       K = INT(MK)
       IF (NGUARD <= 1) NMWA = N1 + 2

!              Subtract MB from MA.

       KP1 = MIN(N1,K+1)
       MWA%MP(K+2) = 0
       DO J = 1, KP1
          MWA%MP(J+1) = MA%MP(J+1)
       ENDDO
       KP2 = K + 2

!              (Inner Loop)

       DO J = KP2+1, N1+1
          MWA%MP(J) = MA%MP(J) - MB%MP(J-K)
       ENDDO

       N2 = QX%NDIG + 2
       IF (N2-K <= 1) N2 = 2 + K
       NK = MIN(NMWA,N1+K)
       DO J = N2, NK
          MWA%MP(J+1) = -MB%MP(J-K+1)
       ENDDO
       NK1 = NK + 1
       DO J = NK1, NMWA
          MWA%MP(J+1) = 0
       ENDDO

!              Normalize.  Fix the sign of any negative digit.

       IF (K > 0) THEN
           DO J = NMWA, KP2, -1
              IF (MWA%MP(J+1) < 0) THEN
                  MWA%MP(J+1) = MWA%MP(J+1) + QX%MBASE
                  MWA%MP(J) = MWA%MP(J) - 1
              ENDIF
           ENDDO

           KPT = KP2 - 1
  110      IF (MWA%MP(KPT+1) < 0 .AND. KPT >= 3) THEN
               MWA%MP(KPT+1) = MWA%MP(KPT+1) + QX%MBASE
               MWA%MP(KPT) = MWA%MP(KPT) - 1
               KPT = KPT - 1
               GO TO 110
           ENDIF
```