

MODULE FM\_DOUBLE\_INT

! FM\_doubleint 1.4 David M. Smith Double Length Integer Support

! This module extends the definition of basic FM types (FM), (IM), and (ZM) so they can interact  
! with double length integer variables.

! Warning: This module is needed only when the user's program explicitly declares double length  
! integer variables. If double length integers are obtained by using a compiler switch  
! to change the default integer size for the entire program (such as with gfortran's  
! -fdefault-integer-8 option), then compiling the basic FM package with the same option  
! means this module is not needed.

! Not all compilers might support double length integers, but for those that do, variables can be  
! declared via the SELECTED\_REAL\_KIND function.

! For example, when this module was first written, typical computer hardware supported 32-bit  
! integers as default precision and 64-bits as double precision. 64-bit integers allowed values  
! up to  $2^{63} - 1$ , which has 19 decimal digits.

! So SELECTED\_INT\_KIND(15) could be used to select this 64-bit format.

! The routines in this interface extend basic functions like TO\_FM, TO\_IM, TO\_ZM so they can be  
! used with quad real or complex arguments. New conversion function TO\_DOUBLE\_INT will take FM,  
! IM, or ZM inputs and convert to double integers.

! Other mixed-mode operations, such as assignment (  $a = b$  ), logical comparisons, and arithmetic  
! are also provided. As with the basic FMZM module, assignments and arithmetic may also involve  
! 1 or 2-dimensional arrays.

USE FMZM

INTEGER, PARAMETER :: DOUBLE\_INT = SELECTED\_INT\_KIND(15)

```
INTERFACE TO_FM
  MODULE PROCEDURE FM_DI
  MODULE PROCEDURE FM_DI1
  MODULE PROCEDURE FM_DI2
END INTERFACE
```

```
INTERFACE TO_IM
  MODULE PROCEDURE IM_DI
  MODULE PROCEDURE IM_DI1
  MODULE PROCEDURE IM_DI2
END INTERFACE
```

```
INTERFACE TO_ZM
  MODULE PROCEDURE ZM_DI
  MODULE PROCEDURE ZM2_DI
  MODULE PROCEDURE ZM_DI1
  MODULE PROCEDURE ZM_DI2
END INTERFACE
```

```
INTERFACE TO_DOUBLE_INT
  MODULE PROCEDURE FM_2DI
  MODULE PROCEDURE IM_2DI
  MODULE PROCEDURE ZM_2DI
END INTERFACE
```

```
MODULE PROCEDURE FM_2DI1
MODULE PROCEDURE IM_2DI1
MODULE PROCEDURE ZM_2DI1
MODULE PROCEDURE FM_2DI2
MODULE PROCEDURE IM_2DI2
MODULE PROCEDURE ZM_2DI2
END INTERFACE
```

```
INTERFACE ASSIGNMENT (=)
MODULE PROCEDURE FMEQ_DIFM
MODULE PROCEDURE FMEQ_DIIM
MODULE PROCEDURE FMEQ_DIZM
MODULE PROCEDURE FMEQ_FMDI
MODULE PROCEDURE FMEQ_IMDI
MODULE PROCEDURE FMEQ_ZMDI
MODULE PROCEDURE FMEQ_FM1DI
MODULE PROCEDURE FMEQ_DI1FM
MODULE PROCEDURE FMEQ_FM1DI1
MODULE PROCEDURE FMEQ_DI1FM1
MODULE PROCEDURE FMEQ_IM1DI
MODULE PROCEDURE FMEQ_DI1IM
MODULE PROCEDURE FMEQ_IM1DI1
MODULE PROCEDURE FMEQ_DI1IM1
MODULE PROCEDURE FMEQ_ZM1DI
MODULE PROCEDURE FMEQ_DI1ZM
MODULE PROCEDURE FMEQ_ZM1DI1
MODULE PROCEDURE FMEQ_DI1ZM1
MODULE PROCEDURE FMEQ_FM2DI
MODULE PROCEDURE FMEQ_DI2FM
MODULE PROCEDURE FMEQ_FM2DI2
MODULE PROCEDURE FMEQ_DI2FM2
MODULE PROCEDURE FMEQ_IM2DI
MODULE PROCEDURE FMEQ_DI2IM
MODULE PROCEDURE FMEQ_IM2DI2
MODULE PROCEDURE FMEQ_DI2IM2
MODULE PROCEDURE FMEQ_ZM2DI
MODULE PROCEDURE FMEQ_DI2ZM
MODULE PROCEDURE FMEQ_ZM2DI2
MODULE PROCEDURE FMEQ_DI2ZM2
END INTERFACE
```

```
INTERFACE OPERATOR (==)
MODULE PROCEDURE FMLEQ_DIFM
MODULE PROCEDURE FMLEQ_DIIM
MODULE PROCEDURE FMLEQ_DIZM
MODULE PROCEDURE FMLEQ_FMDI
MODULE PROCEDURE FMLEQ_IMDI
MODULE PROCEDURE FMLEQ_ZMDI
END INTERFACE
```

```
INTERFACE OPERATOR (/=)
MODULE PROCEDURE FMLNE_DIFM
MODULE PROCEDURE FMLNE_DIIM
MODULE PROCEDURE FMLNE_DIZM
MODULE PROCEDURE FMLNE_FMDI
MODULE PROCEDURE FMLNE_IMDI
MODULE PROCEDURE FMLNE_ZMDI
END INTERFACE
```

```
INTERFACE OPERATOR (>)  
  MODULE PROCEDURE FMLGT_DIFM  
  MODULE PROCEDURE FMLGT_DIIM  
  MODULE PROCEDURE FMLGT_FMDI  
  MODULE PROCEDURE FMLGT_IMDI  
END INTERFACE
```

```
INTERFACE OPERATOR (>=)  
  MODULE PROCEDURE FMLGE_DIFM  
  MODULE PROCEDURE FMLGE_DIIM  
  MODULE PROCEDURE FMLGE_FMDI  
  MODULE PROCEDURE FMLGE_IMDI  
END INTERFACE
```

```
INTERFACE OPERATOR (<)  
  MODULE PROCEDURE FMLLT_DIFM  
  MODULE PROCEDURE FMLLT_DIIM  
  MODULE PROCEDURE FMLLT_FMDI  
  MODULE PROCEDURE FMLLT_IMDI  
END INTERFACE
```

```
INTERFACE OPERATOR (<=)  
  MODULE PROCEDURE FMLLE_DIFM  
  MODULE PROCEDURE FMLLE_DIIM  
  MODULE PROCEDURE FMLLE_FMDI  
  MODULE PROCEDURE FMLLE_IMDI  
END INTERFACE
```

```
INTERFACE OPERATOR (+)  
  MODULE PROCEDURE FMADD_DIFM  
  MODULE PROCEDURE FMADD_DIIM  
  MODULE PROCEDURE FMADD_DIZM  
  MODULE PROCEDURE FMADD_FMDI  
  MODULE PROCEDURE FMADD_IMDI  
  MODULE PROCEDURE FMADD_ZMDI  
  MODULE PROCEDURE FMADD_DIFM1  
  MODULE PROCEDURE FMADD_DIIM1  
  MODULE PROCEDURE FMADD_FMDI1  
  MODULE PROCEDURE FMADD_FM1DI  
  MODULE PROCEDURE FMADD_DI1FM  
  MODULE PROCEDURE FMADD_DI1FM1  
  MODULE PROCEDURE FMADD_FM1DI1  
  MODULE PROCEDURE FMADD_IMDI1  
  MODULE PROCEDURE FMADD_IM1DI  
  MODULE PROCEDURE FMADD_DI1IM  
  MODULE PROCEDURE FMADD_DI1IM1  
  MODULE PROCEDURE FMADD_IM1DI1  
  MODULE PROCEDURE FMADD_DIZM1  
  MODULE PROCEDURE FMADD_ZMDI1  
  MODULE PROCEDURE FMADD_ZM1DI  
  MODULE PROCEDURE FMADD_DI1ZM  
  MODULE PROCEDURE FMADD_DI1ZM1  
  MODULE PROCEDURE FMADD_ZM1DI1  
  MODULE PROCEDURE FMADD_DIFM2  
  MODULE PROCEDURE FMADD_DIIM2  
  MODULE PROCEDURE FMADD_FMDI2  
  MODULE PROCEDURE FMADD_FM2DI
```

```
MODULE PROCEDURE FMADD_DI2FM
MODULE PROCEDURE FMADD_DI2FM2
MODULE PROCEDURE FMADD_FM2DI2
MODULE PROCEDURE FMADD_IMDI2
MODULE PROCEDURE FMADD_IM2DI
MODULE PROCEDURE FMADD_DI2IM
MODULE PROCEDURE FMADD_DI2IM2
MODULE PROCEDURE FMADD_IM2DI2
MODULE PROCEDURE FMADD_DIZM2
MODULE PROCEDURE FMADD_ZMDI2
MODULE PROCEDURE FMADD_ZM2DI
MODULE PROCEDURE FMADD_DI2ZM
MODULE PROCEDURE FMADD_DI2ZM2
MODULE PROCEDURE FMADD_ZM2DI2
END INTERFACE
```

```
INTERFACE OPERATOR (-)
```

```
MODULE PROCEDURE FMSUB_DIFM
MODULE PROCEDURE FMSUB_DIIM
MODULE PROCEDURE FMSUB_DIZM
MODULE PROCEDURE FMSUB_FMDI
MODULE PROCEDURE FMSUB_IMDI
MODULE PROCEDURE FMSUB_ZMDI
MODULE PROCEDURE FMSUB_DIFM1
MODULE PROCEDURE FMSUB_DIIM1
MODULE PROCEDURE FMSUB_FMDI1
MODULE PROCEDURE FMSUB_FM1DI
MODULE PROCEDURE FMSUB_DI1FM
MODULE PROCEDURE FMSUB_DI1FM1
MODULE PROCEDURE FMSUB_FM1DI1
MODULE PROCEDURE FMSUB_IMDI1
MODULE PROCEDURE FMSUB_IM1DI
MODULE PROCEDURE FMSUB_DI1IM
MODULE PROCEDURE FMSUB_DI1IM1
MODULE PROCEDURE FMSUB_IM1DI1
MODULE PROCEDURE FMSUB_DIZM1
MODULE PROCEDURE FMSUB_ZMDI1
MODULE PROCEDURE FMSUB_ZM1DI
MODULE PROCEDURE FMSUB_DI1ZM
MODULE PROCEDURE FMSUB_DI1ZM1
MODULE PROCEDURE FMSUB_ZM1DI1
MODULE PROCEDURE FMSUB_DIFM2
MODULE PROCEDURE FMSUB_DIIM2
MODULE PROCEDURE FMSUB_FMDI2
MODULE PROCEDURE FMSUB_FM2DI
MODULE PROCEDURE FMSUB_DI2FM
MODULE PROCEDURE FMSUB_DI2FM2
MODULE PROCEDURE FMSUB_FM2DI2
MODULE PROCEDURE FMSUB_IMDI2
MODULE PROCEDURE FMSUB_IM2DI
MODULE PROCEDURE FMSUB_DI2IM
MODULE PROCEDURE FMSUB_DI2IM2
MODULE PROCEDURE FMSUB_IM2DI2
MODULE PROCEDURE FMSUB_DIZM2
MODULE PROCEDURE FMSUB_ZMDI2
MODULE PROCEDURE FMSUB_ZM2DI
MODULE PROCEDURE FMSUB_DI2ZM
MODULE PROCEDURE FMSUB_DI2ZM2
```

```
MODULE PROCEDURE FMSUB_ZM2DI2
END INTERFACE
```

```
INTERFACE OPERATOR (*)
```

```
MODULE PROCEDURE FMMPY_DIFM
MODULE PROCEDURE FMMPY_DIIM
MODULE PROCEDURE FMMPY_DIZM
MODULE PROCEDURE FMMPY_FMDI
MODULE PROCEDURE FMMPY_IMDI
MODULE PROCEDURE FMMPY_ZMDI
MODULE PROCEDURE FMMPY_DIFM1
MODULE PROCEDURE FMMPY_DIIM1
MODULE PROCEDURE FMMPY_FMDI1
MODULE PROCEDURE FMMPY_FM1DI
MODULE PROCEDURE FMMPY_DI1FM
MODULE PROCEDURE FMMPY_DI1FM1
MODULE PROCEDURE FMMPY_FM1DI1
MODULE PROCEDURE FMMPY_IMDI1
MODULE PROCEDURE FMMPY_IM1DI
MODULE PROCEDURE FMMPY_DI1IM
MODULE PROCEDURE FMMPY_DI1IM1
MODULE PROCEDURE FMMPY_IM1DI1
MODULE PROCEDURE FMMPY_DIZM1
MODULE PROCEDURE FMMPY_ZMDI1
MODULE PROCEDURE FMMPY_ZM1DI
MODULE PROCEDURE FMMPY_DI1ZM
MODULE PROCEDURE FMMPY_DI1ZM1
MODULE PROCEDURE FMMPY_ZM1DI1
MODULE PROCEDURE FMMPY_DIFM2
MODULE PROCEDURE FMMPY_DIIM2
MODULE PROCEDURE FMMPY_FMDI2
MODULE PROCEDURE FMMPY_FM2DI
MODULE PROCEDURE FMMPY_DI2FM
MODULE PROCEDURE FMMPY_DI2FM2
MODULE PROCEDURE FMMPY_FM2DI2
MODULE PROCEDURE FMMPY_IMDI2
MODULE PROCEDURE FMMPY_IM2DI
MODULE PROCEDURE FMMPY_DI2IM
MODULE PROCEDURE FMMPY_DI2IM2
MODULE PROCEDURE FMMPY_IM2DI2
MODULE PROCEDURE FMMPY_DIZM2
MODULE PROCEDURE FMMPY_ZMDI2
MODULE PROCEDURE FMMPY_ZM2DI
MODULE PROCEDURE FMMPY_DI2ZM
MODULE PROCEDURE FMMPY_DI2ZM2
MODULE PROCEDURE FMMPY_ZM2DI2
```

```
END INTERFACE
```

```
INTERFACE OPERATOR (/)
```

```
MODULE PROCEDURE FMDIV_DIFM
MODULE PROCEDURE FMDIV_DIIM
MODULE PROCEDURE FMDIV_DIZM
MODULE PROCEDURE FMDIV_FMDI
MODULE PROCEDURE FMDIV_IMDI
MODULE PROCEDURE FMDIV_ZMDI
MODULE PROCEDURE FMDIV_DIFM1
MODULE PROCEDURE FMDIV_DIIM1
MODULE PROCEDURE FMDIV_FMDI1
```

```

MODULE PROCEDURE FMDIV_FM1DI
MODULE PROCEDURE FMDIV_DI1FM
MODULE PROCEDURE FMDIV_DI1FM1
MODULE PROCEDURE FMDIV_FM1DI1
MODULE PROCEDURE FMDIV_IMDI1
MODULE PROCEDURE FMDIV_IM1DI
MODULE PROCEDURE FMDIV_DI1IM
MODULE PROCEDURE FMDIV_DI1IM1
MODULE PROCEDURE FMDIV_IM1DI1
MODULE PROCEDURE FMDIV_DIZM1
MODULE PROCEDURE FMDIV_ZMDI1
MODULE PROCEDURE FMDIV_ZM1DI
MODULE PROCEDURE FMDIV_DI1ZM
MODULE PROCEDURE FMDIV_DI1ZM1
MODULE PROCEDURE FMDIV_ZM1DI1
MODULE PROCEDURE FMDIV_DIFM2
MODULE PROCEDURE FMDIV_DIIM2
MODULE PROCEDURE FMDIV_FMDI2
MODULE PROCEDURE FMDIV_FM2DI
MODULE PROCEDURE FMDIV_DI2FM
MODULE PROCEDURE FMDIV_DI2FM2
MODULE PROCEDURE FMDIV_FM2DI2
MODULE PROCEDURE FMDIV_IMDI2
MODULE PROCEDURE FMDIV_IM2DI
MODULE PROCEDURE FMDIV_DI2IM
MODULE PROCEDURE FMDIV_DI2IM2
MODULE PROCEDURE FMDIV_IM2DI2
MODULE PROCEDURE FMDIV_DIZM2
MODULE PROCEDURE FMDIV_ZMDI2
MODULE PROCEDURE FMDIV_ZM2DI
MODULE PROCEDURE FMDIV_DI2ZM
MODULE PROCEDURE FMDIV_DI2ZM2
MODULE PROCEDURE FMDIV_ZM2DI2
END INTERFACE

```

```

INTERFACE OPERATOR (**)
  MODULE PROCEDURE FMPWR_DIFM
  MODULE PROCEDURE FMPWR_DIIM
  MODULE PROCEDURE FMPWR_DIZM
  MODULE PROCEDURE FMPWR_FMDI
  MODULE PROCEDURE FMPWR_IMDI
  MODULE PROCEDURE FMPWR_ZMDI
END INTERFACE

```

CONTAINS

```

SUBROUTINE FMDI2M(IVAL,MA)

```

! Convert double length integer IVAL to multiple precision MA.

```

USE FMVALS
IMPLICIT NONE

```

```

TYPE(MULTI) :: MA
INTEGER (DOUBLE_INT) :: IVAL

```

```

REAL (KIND(1.0D0)) :: MK,ML,MVAL
INTEGER (DOUBLE_INT) :: J,JM2,KB,KB1,N1,NMVAL,NV2

```

```
CHARACTER(50) :: STR
INTENT (IN) :: IVAL
INTENT (INOUT) :: MA
```

```
IF (.NOT. ALLOCATED(MA%MP)) THEN
  ALLOCATE(MA%MP(NDIG+2),STAT=K_STAT)
  IF (K_STAT /= 0) CALL FMDEFINE_ERROR
ELSE IF (SIZE(MA%MP) < NDIG+2) THEN
  DEALLOCATE(MA%MP)
  ALLOCATE(MA%MP(NDIG+2),STAT=K_STAT)
  IF (K_STAT /= 0) CALL FMDEFINE_ERROR
ENDIF
```

```
IF (MBLOGS /= MBASE) CALL FMCONS
KFLAG = 0
N1 = NDIG + 1
```

```
IF (ABS(IVAL) > MXBASE) THEN
  WRITE (STR, "(I50)") IVAL
  CALL FMST2M(STR,MA)
  GO TO 150
ELSE
  MVAL = ABS(IVAL)
  NMVAL = MVAL
  NV2 = NMVAL - 1
  IF (NMVAL /= ABS(IVAL) .OR. NV2 /= ABS(IVAL)-1) THEN
    WRITE (STR, "(I50)") IVAL
    CALL FMST2M(STR,MA)
    GO TO 150
  ENDIF
ENDIF
```

! Check for small IVAL.

```
IF (MVAL < MBASE) THEN
  DO J = 3, N1
    MA%MP(J+1) = 0
  ENDDO
  IF (IVAL >= 0) THEN
    MA%MP(3) = IVAL
    MA%MP(1) = 1
  ELSE
    MA%MP(3) = -IVAL
    MA%MP(1) = -1
  ENDIF
  IF (IVAL == 0) THEN
    MA%MP(2) = 0
  ELSE
    MA%MP(2) = 1
  ENDIF
  GO TO 150
ENDIF
```

! Compute and store the digits, right to left.

```
MA%MP(2) = 0
J = NDIG + 1
```

```

120 MK = AINT (MVAL/MBASE)
ML = MVAL - MK*MBASE
MA%MP(2) = MA%MP(2) + 1
MA%MP(J+1) = ML
IF (MK > 0) THEN
    MVAL = MK
    J = J - 1
    IF (J >= 2) GO TO 120

```

! Here IVAL cannot be expressed exactly.

```

WRITE (STR,"(I50)") IVAL
CALL FMST2M(STR,MA)
RETURN
ENDIF

```

! Normalize MA.

```

KB = N1 - J + 2
JM2 = J - 2
DO J = 2, KB
    MA%MP(J+1) = MA%MP(J+JM2+1)
ENDDO
KB1 = KB + 1
IF (KB1 <= N1) THEN
    DO J = KB1, N1
        MA%MP(J+1) = 0
    ENDDO
ENDIF

```

```

MA%MP(1) = 1
IF (IVAL < 0 .AND. MA%MP(2) /= MUNKNO .AND. MA%MP(3) /= 0) MA%MP(1) = -1

```

```

150 RETURN
END SUBROUTINE FMDI2M

```

```

SUBROUTINE FMM2DI(MA,IVAL)

```

! Convert multiple precision MA to double length integer IVAL.

```

USE FMVALS
IMPLICIT NONE

```

```

TYPE(MULTI) :: MA
INTEGER (DOUBLE_INT) :: IVAL

```

```

INTEGER (DOUBLE_INT) :: IBASE, J, KA, KB, LARGE, N1
INTENT (IN) :: MA
INTENT (INOUT) :: IVAL

```

```

KFLAG = 0
N1 = NDIG + 1
LARGE = HUGE(IVAL)/MBASE
IBASE = MBASE
IVAL = 0
IF (MA%MP(2) <= 0) THEN
    IF (MA%MP(3) /= 0) KFLAG = 2

```



```

    RETURN
ENDIF

KB = MA%MP(2) + 1
IVAL = ABS(MA%MP(3))
IF (KB >= 3) THEN
    DO J = 3, KB
        IF (IVAL > LARGE) THEN
            KFLAG = -4
            IF (MA%MP(2) /= MUNKNO) CALL FMWARN
            IVAL = IUNKNO
            RETURN
        ENDIF
        IF (J <= N1) THEN
            IVAL = IVAL*IBASE
            IF (IVAL > HUGE(IVAL)-MA%MP(J+1)) THEN
                KFLAG = -4
                IF (MA%MP(2) /= MUNKNO) CALL FMWARN
                IVAL = IUNKNO
                RETURN
            ELSE
                IVAL = IVAL + INT(MA%MP(J+1))
            ENDIF
        ELSE
            IVAL = IVAL*IBASE
        ENDIF
    ENDDO
ENDIF

IF (MA%MP(1) < 0) IVAL = -IVAL

```

! Check to see if MA is an integer.

```

KA = KB + 1
IF (KA <= N1) THEN
    DO J = KA, N1
        IF (MA%MP(J+1) /= 0) THEN
            KFLAG = 2
            RETURN
        ENDIF
    ENDDO
ENDIF

RETURN
END SUBROUTINE FMM2DI

```

```

SUBROUTINE IMDI2M(IVAL,MA)

```

! MA = IVAL

! Convert a double length integer to an IM number.

```

USE FMVALS
IMPLICIT NONE

TYPE(MULTI) :: MA
INTEGER (DOUBLE_INT) :: IVAL

```

```

INTEGER :: NDSAVE
INTENT (IN) :: IVAL
INTENT (INOUT) :: MA
TYPE(MULTI), SAVE :: MTLVFM

CALL FMDI2M(IVAL,MTLVFM)

IF (INT(MTLVFM%MP(2)) > NDIG) THEN
  NDSAVE = NDIG
  NDIG = MAX(2,INT(MTLVFM%MP(2)))
  CALL FMDI2M(IVAL,MTLVFM)
  CALL IMFM2I(MTLVFM,MA)
  NDIG = NDSAVE
ELSE
  CALL IMFM2I(MTLVFM,MA)
ENDIF

RETURN
END SUBROUTINE IMDI2M

```

```

SUBROUTINE IMM2DI(MA,IVAL)

```

! IVAL = MA

! Convert an IM number to double length integer.

```

USE FMVALS
IMPLICIT NONE

TYPE(MULTI) :: MA
INTEGER (DOUBLE_INT) :: IVAL

INTEGER :: NDSAVE
INTENT (IN) :: MA
INTENT (INOUT) :: IVAL
TYPE(MULTI), SAVE :: MTLVFM

NDSAVE = NDIG
NDIG = MAX(2,INT(MA%MP(2)))

CALL IMI2FM(MA,MTLVFM)
CALL FMM2DI(MTLVFM,IVAL)

NDIG = NDSAVE
RETURN
END SUBROUTINE IMM2DI

```

```

FUNCTION FM_DI(D)      RESULT (RETURN_VALUE)
  USE FMVALS
  IMPLICIT NONE
  TYPE (FM) :: RETURN_VALUE
  INTEGER (DOUBLE_INT) :: D
  INTENT (IN) :: D
  CALL FMDI2M(D,RETURN_VALUE%FM)
END FUNCTION FM_DI

```

```

FUNCTION FM_DI1(D)      RESULT (RETURN_VALUE)
  USE FMVALS
  IMPLICIT NONE
  INTEGER (DOUBLE_INT), DIMENSION(:) :: D
  TYPE (FM), DIMENSION(SIZE(D)) :: RETURN_VALUE
  INTEGER :: J,N
  INTENT (IN) :: D
  N = SIZE(D)
  DO J = 1, N
    CALL FMDI2M(D(J),RETURN_VALUE(J)%MFM)
  ENDDO
END FUNCTION FM_DI1

```

```

FUNCTION FM_DI2(D)      RESULT (RETURN_VALUE)
  USE FMVALS
  IMPLICIT NONE
  INTEGER (DOUBLE_INT), DIMENSION(:, :) :: D
  TYPE (FM), DIMENSION(SIZE(D,DIM=1),SIZE(D,DIM=2)) :: RETURN_VALUE
  INTEGER :: J,K
  INTENT (IN) :: D
  DO J = 1, SIZE(D,DIM=1)
    DO K = 1, SIZE(D,DIM=2)
      CALL FMDI2M(D(J,K),RETURN_VALUE(J,K)%MFM)
    ENDDO
  ENDDO
END FUNCTION FM_DI2

```

```

FUNCTION IM_DI(D)      RESULT (RETURN_VALUE)
  USE FMVALS
  IMPLICIT NONE
  TYPE (IM) :: RETURN_VALUE
  INTEGER (DOUBLE_INT) :: D
  INTENT (IN) :: D
  CALL IMDI2M(D,RETURN_VALUE%MIM)
END FUNCTION IM_DI

```

```

FUNCTION IM_DI1(D)      RESULT (RETURN_VALUE)
  USE FMVALS
  IMPLICIT NONE
  INTEGER (DOUBLE_INT), DIMENSION(:) :: D
  TYPE (IM), DIMENSION(SIZE(D)) :: RETURN_VALUE
  INTEGER :: J,N
  INTENT (IN) :: D
  N = SIZE(D)
  DO J = 1, N
    CALL IMDI2M(D(J),RETURN_VALUE(J)%MIM)
  ENDDO
END FUNCTION IM_DI1

```

```

FUNCTION IM_DI2(D)      RESULT (RETURN_VALUE)
  USE FMVALS
  IMPLICIT NONE
  INTEGER (DOUBLE_INT), DIMENSION(:, :) :: D
  TYPE (IM), DIMENSION(SIZE(D,DIM=1),SIZE(D,DIM=2)) :: RETURN_VALUE
  INTEGER :: J,K
  INTENT (IN) :: D
  DO J = 1, SIZE(D,DIM=1)

```

```

      DO K = 1, SIZE(D,DIM=2)
        CALL IMDI2M(D(J,K),RETURN_VALUE(J,K)%MIM)
      ENDDO
    ENDDO
  END FUNCTION IM_DI2

FUNCTION ZM_DI(D)      RESULT (RETURN_VALUE)
  USE FMVALS
  IMPLICIT NONE
  TYPE (ZM) :: RETURN_VALUE
  INTEGER (DOUBLE_INT) :: D
  INTENT (IN) :: D
  TYPE(MULTI), SAVE :: MTLVFM,MULVFM
  CALL FMDI2M(D,MTLVFM)
  CALL FMI2M(0,MULVFM)
  CALL ZMCMPX(MTLVFM,MULVFM,RETURN_VALUE%MZM)
END FUNCTION ZM_DI

FUNCTION ZM2_DI(D1,D2)      RESULT (RETURN_VALUE)
  USE FMVALS
  IMPLICIT NONE
  TYPE (ZM) :: RETURN_VALUE
  INTEGER (DOUBLE_INT) :: D1,D2
  INTENT (IN) :: D1,D2
  TYPE(MULTI), SAVE :: MTLVFM,MULVFM
  CALL FMDI2M(D1,MTLVFM)
  CALL FMDI2M(D2,MULVFM)
  CALL ZMCMPX(MTLVFM,MULVFM,RETURN_VALUE%MZM)
END FUNCTION ZM2_DI

FUNCTION ZM_DI1(D)      RESULT (RETURN_VALUE)
  USE FMVALS
  IMPLICIT NONE
  INTEGER (DOUBLE_INT), DIMENSION(:) :: D
  TYPE (ZM), DIMENSION(SIZE(D)) :: RETURN_VALUE
  INTEGER :: J,N
  INTENT (IN) :: D
  TYPE(MULTI), SAVE :: MTLVFM,MULVFM
  N = SIZE(D)
  CALL FMI2M(0,MULVFM)
  DO J = 1, N
    CALL FMDI2M(D(J),MTLVFM)
    CALL ZMCMPX(MTLVFM,MULVFM,RETURN_VALUE(J)%MZM)
  ENDDO
END FUNCTION ZM_DI1

FUNCTION ZM_DI2(D)      RESULT (RETURN_VALUE)
  USE FMVALS
  IMPLICIT NONE
  INTEGER (DOUBLE_INT), DIMENSION(:,:) :: D
  TYPE (ZM), DIMENSION(SIZE(D,DIM=1),SIZE(D,DIM=2)) :: RETURN_VALUE
  INTEGER :: J,K
  INTENT (IN) :: D
  TYPE(MULTI), SAVE :: MTLVFM,MULVFM
  CALL FMI2M(0,MULVFM)
  DO J = 1, SIZE(D,DIM=1)
    DO K = 1, SIZE(D,DIM=2)
      CALL FMDI2M(D(J,K),MTLVFM)
    
```

```

        CALL ZMCOMPX(MTLVFM,MULVFM,RETURN_VALUE(J,K)%MZM)
    ENDDO
ENDDO
END FUNCTION ZM_DI2

FUNCTION FM_2DI(MA)      RESULT (RETURN_VALUE)
    USE FMVALS
    IMPLICIT NONE
    TYPE (FM) :: MA
    INTEGER (DOUBLE_INT) :: RETURN_VALUE
    INTENT (IN) :: MA
    CALL FM_UNDEF_INP(MA)
    CALL FMM2DI(MA%FM,RETURN_VALUE)
END FUNCTION FM_2DI

FUNCTION IM_2DI(MA)      RESULT (RETURN_VALUE)
    USE FMVALS
    IMPLICIT NONE
    TYPE (IM) :: MA
    INTEGER (DOUBLE_INT) :: RETURN_VALUE
    INTENT (IN) :: MA
    CALL FM_UNDEF_INP(MA)
    CALL IMM2DI(MA%MIM,RETURN_VALUE)
END FUNCTION IM_2DI

FUNCTION ZM_2DI(MA)      RESULT (RETURN_VALUE)
    USE FMVALS
    IMPLICIT NONE
    TYPE (ZM) :: MA
    INTEGER (DOUBLE_INT) :: RETURN_VALUE
    INTENT (IN) :: MA
    TYPE(MULTI), SAVE :: MTLVFM
    CALL FM_UNDEF_INP(MA)
    CALL ZMREAL(MA%MZM,MTLVFM)
    CALL FMM2DI(MTLVFM,RETURN_VALUE)
END FUNCTION ZM_2DI

FUNCTION FM_2DI1(MA)      RESULT (RETURN_VALUE)
    USE FMVALS
    IMPLICIT NONE
    TYPE (FM), DIMENSION(:) :: MA
    INTEGER (DOUBLE_INT), DIMENSION(SIZE(MA)) :: RETURN_VALUE
    INTEGER :: J,N
    INTENT (IN) :: MA
    CALL FM_UNDEF_INP(MA)
    N = SIZE(MA)
    DO J = 1, N
        CALL FMM2DI(MA(J)%FM,RETURN_VALUE(J))
    ENDDO
END FUNCTION FM_2DI1

FUNCTION IM_2DI1(MA)      RESULT (RETURN_VALUE)
    USE FMVALS
    IMPLICIT NONE
    TYPE (IM), DIMENSION(:) :: MA
    INTEGER (DOUBLE_INT), DIMENSION(SIZE(MA)) :: RETURN_VALUE
    INTEGER :: J,N
    INTENT (IN) :: MA

```

```

CALL FM_UNDEF_INP(MA)
N = SIZE(MA)
DO J = 1, N
    CALL IMM2DI(MA(J)%MIM, RETURN_VALUE(J))
ENDDO
END FUNCTION IM_2DI1

```

```

FUNCTION ZM_2DI1(MA)      RESULT (RETURN_VALUE)
    USE FMVALS
    IMPLICIT NONE
    TYPE (ZM), DIMENSION(:) :: MA
    INTEGER (DOUBLE_INT), DIMENSION(SIZE(MA)) :: RETURN_VALUE
    INTEGER :: J, N
    INTENT (IN) :: MA
    CALL FM_UNDEF_INP(MA)
    N = SIZE(MA)
    DO J = 1, N
        CALL FMM2DI(MA(J)%MZM(1), RETURN_VALUE(J))
    ENDDO
END FUNCTION ZM_2DI1

```

```

FUNCTION FM_2DI2(MA)      RESULT (RETURN_VALUE)
    USE FMVALS
    IMPLICIT NONE
    TYPE (FM), DIMENSION(:, :) :: MA
    INTEGER (DOUBLE_INT), DIMENSION(SIZE(MA, DIM=1), SIZE(MA, DIM=2)) :: RETURN_VALUE
    INTEGER :: J, K
    INTENT (IN) :: MA
    CALL FM_UNDEF_INP(MA)
    DO J = 1, SIZE(MA, DIM=1)
        DO K = 1, SIZE(MA, DIM=2)
            CALL FMM2DI(MA(J, K)%MFM, RETURN_VALUE(J, K))
        ENDDO
    ENDDO
END FUNCTION FM_2DI2

```

```

FUNCTION IM_2DI2(MA)      RESULT (RETURN_VALUE)
    USE FMVALS
    IMPLICIT NONE
    TYPE (IM), DIMENSION(:, :) :: MA
    INTEGER (DOUBLE_INT), DIMENSION(SIZE(MA, DIM=1), SIZE(MA, DIM=2)) :: RETURN_VALUE
    INTEGER :: J, K
    INTENT (IN) :: MA
    CALL FM_UNDEF_INP(MA)
    DO J = 1, SIZE(MA, DIM=1)
        DO K = 1, SIZE(MA, DIM=2)
            CALL IMM2DI(MA(J, K)%MIM, RETURN_VALUE(J, K))
        ENDDO
    ENDDO
END FUNCTION IM_2DI2

```

```

FUNCTION ZM_2DI2(MA)      RESULT (RETURN_VALUE)
    USE FMVALS
    IMPLICIT NONE
    TYPE (ZM), DIMENSION(:, :) :: MA
    INTEGER (DOUBLE_INT), DIMENSION(SIZE(MA, DIM=1), SIZE(MA, DIM=2)) :: RETURN_VALUE
    INTEGER :: J, K
    INTENT (IN) :: MA

```

```

CALL FM_UNDEF_INP(MA)
DO J = 1, SIZE(MA,DIM=1)
  DO K = 1, SIZE(MA,DIM=2)
    CALL FMM2DI(MA(J,K)%MZM(1),RETURN_VALUE(J,K))
  ENDDO
ENDDO
END FUNCTION ZM_2DI2

```

```

SUBROUTINE FMEQ_DIFM(D,MA)
USE FMVALS
IMPLICIT NONE
TYPE (FM) :: MA
INTEGER (DOUBLE_INT) :: D
INTENT (INOUT) :: D
INTENT (IN) :: MA
CALL FM_UNDEF_INP(MA)
CALL FMM2DI(MA%MFM,D)
END SUBROUTINE FMEQ_DIFM

```

```

SUBROUTINE FMEQ_DIIM(D,MA)
USE FMVALS
IMPLICIT NONE
TYPE (IM) :: MA
INTEGER (DOUBLE_INT) :: D
INTENT (INOUT) :: D
INTENT (IN) :: MA
CALL FM_UNDEF_INP(MA)
CALL IMM2DI(MA%MIM,D)
END SUBROUTINE FMEQ_DIIM

```

```

SUBROUTINE FMEQ_DIZM(D,MA)
USE FMVALS
IMPLICIT NONE
TYPE (ZM) :: MA
INTEGER (DOUBLE_INT) :: D
INTENT (INOUT) :: D
INTENT (IN) :: MA
TYPE(MULTI), SAVE :: MTLVFM
CALL FM_UNDEF_INP(MA)
CALL ZMREAL(MA%MZM,MTLVFM)
CALL FMM2DI(MTLVFM,D)
END SUBROUTINE FMEQ_DIZM

```

```

SUBROUTINE FMEQ_FMDI(MA,D)
USE FMVALS
IMPLICIT NONE
TYPE (FM) :: MA
INTEGER (DOUBLE_INT) :: D
INTENT (INOUT) :: MA
INTENT (IN) :: D
CALL FMDI2M(D,MA%MFM)
END SUBROUTINE FMEQ_FMDI

```

```

SUBROUTINE FMEQ_IMDI(MA,D)
USE FMVALS
IMPLICIT NONE
TYPE (IM) :: MA

```

```

INTEGER :: IVAL
INTEGER (DOUBLE_INT) :: D
CHARACTER(50) :: ST
INTENT (INOUT) :: MA
INTENT (IN) :: D
IF (ABS(D) < HUGE(1)) THEN
    IVAL = INT(D)
    CALL IMI2M(IVAL,MA%MIM)
ELSE
    WRITE (ST,'(I50)') D
    CALL IMST2M(ST,MA%MIM)
ENDIF
END SUBROUTINE FMEQ_IMDI

```

```

SUBROUTINE FMEQ_ZMDI(MA,D)
USE FMVALS
IMPLICIT NONE
TYPE (ZM) :: MA
INTEGER (DOUBLE_INT) :: D
INTENT (INOUT) :: MA
INTENT (IN) :: D
TYPE(MULTI), SAVE :: MTLVFM,MULVFM
CALL FMDI2M(D,MTLVFM)
CALL FMI2M(0,MULVFM)
CALL ZMCPX(MTLVFM,MULVFM,MA%MZM)
END SUBROUTINE FMEQ_ZMDI

```

```

SUBROUTINE FMEQ_FM1DI(MA,D)
USE FMVALS
IMPLICIT NONE
TYPE (FM), DIMENSION(:) :: MA
INTEGER :: J,N
INTEGER (DOUBLE_INT) :: D
INTENT (INOUT) :: MA
INTENT (IN) :: D
TYPE(MULTI), SAVE :: MTLVFM
N = SIZE(MA)
CALL FMDI2M(D,MTLVFM)
DO J = 1, N
    CALL FMEQ(MTLVFM,MA(J)%MFM)
ENDDO
END SUBROUTINE FMEQ_FM1DI

```

```

SUBROUTINE FMEQ_DI1FM(D,MA)
USE FMVALS
IMPLICIT NONE
TYPE (FM) :: MA
INTEGER (DOUBLE_INT), DIMENSION(:) :: D
INTEGER (DOUBLE_INT) :: D2
INTEGER :: J,N
INTENT (INOUT) :: D
INTENT (IN) :: MA
CALL FM_UNDEF_INP(MA)
N = SIZE(D)
CALL FMM2DI(MA%MFM,D2)
DO J = 1, N
    D(J) = D2
ENDDO

```



```
END SUBROUTINE FMEQ_DI1FM
```

```
SUBROUTINE FMEQ_FM1DI1(MA,D)
  USE FMVALS
  IMPLICIT NONE
  TYPE (FM), DIMENSION(:) :: MA
  INTEGER :: J,N
  INTEGER (DOUBLE_INT), DIMENSION(:) :: D
  INTENT (INOUT) :: MA
  INTENT (IN) :: D
  TYPE(MULTI), SAVE :: MTLVFM
  IF (SIZE(MA) /= SIZE(D)) THEN
    CALL FMST2M(' UNKNOWN ',MTLVFM)
    DO J = 1, SIZE(MA)
      CALL FMEQ(MTLVFM,MA(J)%MFM)
    ENDDO
    RETURN
  ENDIF
  N = SIZE(MA)
  DO J = 1, N
    CALL FMDI2M(D(J),MA(J)%MFM)
  ENDDO
END SUBROUTINE FMEQ_FM1DI1
```

```
SUBROUTINE FMEQ_DI1FM1(D,MA)
  USE FMVALS
  IMPLICIT NONE
  TYPE (FM), DIMENSION(:) :: MA
  INTEGER (DOUBLE_INT), DIMENSION(:) :: D
  INTEGER :: J,N
  INTENT (INOUT) :: D
  INTENT (IN) :: MA
  CALL FM_UNDEF_INP(MA)
  IF (SIZE(MA) /= SIZE(D)) THEN
    DO J = 1, SIZE(D)
      D(J) = RUNKNO
    ENDDO
    RETURN
  ENDIF
  N = SIZE(D)
  DO J = 1, N
    CALL FMM2DI(MA(J)%MFM,D(J))
  ENDDO
END SUBROUTINE FMEQ_DI1FM1
```

```
SUBROUTINE FMEQ_FM2DI(MA,D)
  USE FMVALS
  IMPLICIT NONE
  TYPE (FM), DIMENSION(:, :) :: MA
  INTEGER :: J,K
  INTEGER (DOUBLE_INT) :: D
  INTENT (INOUT) :: MA
  INTENT (IN) :: D
  TYPE(MULTI), SAVE :: MTLVFM
  CALL FMDI2M(D,MTLVFM)
  DO J = 1, SIZE(MA,DIM=1)
    DO K = 1, SIZE(MA,DIM=2)
      CALL FMEQ(MTLVFM,MA(J,K)%MFM)
    
```

```

        ENDDO
    ENDDO
END SUBROUTINE FMEQ_FM2DI

```

```

SUBROUTINE FMEQ_DI2FM(D,MA)
    USE FMVALS
    IMPLICIT NONE
    TYPE (FM) :: MA
    INTEGER (DOUBLE_INT), DIMENSION(:,,:) :: D
    INTEGER (DOUBLE_INT) :: D2
    INTEGER :: J,K
    INTENT (INOUT) :: D
    INTENT (IN) :: MA
    CALL FM_UNDEF_INP(MA)
    CALL FMM2DI(MA%MFM,D2)
    DO J = 1, SIZE(D,DIM=1)
        DO K = 1, SIZE(D,DIM=2)
            D(J,K) = D2
        ENDDO
    ENDDO
END SUBROUTINE FMEQ_DI2FM

```

```

SUBROUTINE FMEQ_FM2DI2(MA,D)
    USE FMVALS
    IMPLICIT NONE
    TYPE (FM), DIMENSION(:,,:) :: MA
    INTEGER :: J,K
    INTEGER (DOUBLE_INT), DIMENSION(:,,:) :: D
    INTENT (INOUT) :: MA
    INTENT (IN) :: D
    TYPE(MULTI), SAVE :: MTLVFM
    IF (SIZE(MA,DIM=1) /= SIZE(D,DIM=1) .OR. SIZE(MA,DIM=2) /= SIZE(D,DIM=2)) THEN
        CALL FMST2M(' UNKNOWN ',MTLVFM)
        DO J = 1, SIZE(MA,DIM=1)
            DO K = 1, SIZE(MA,DIM=2)
                CALL FMEQ(MTLVFM,MA(J,K)%MFM)
            ENDDO
        ENDDO
        RETURN
    ENDIF
    DO J = 1, SIZE(MA,DIM=1)
        DO K = 1, SIZE(MA,DIM=2)
            CALL FMDI2M(D(J,K),MA(J,K)%MFM)
        ENDDO
    ENDDO
END SUBROUTINE FMEQ_FM2DI2

```

```

SUBROUTINE FMEQ_DI2FM2(D,MA)
    USE FMVALS
    IMPLICIT NONE
    TYPE (FM), DIMENSION(:,,:) :: MA
    INTEGER (DOUBLE_INT), DIMENSION(:,,:) :: D
    INTEGER :: J,K
    INTENT (INOUT) :: D
    INTENT (IN) :: MA
    CALL FM_UNDEF_INP(MA)
    IF (SIZE(MA,DIM=1) /= SIZE(D,DIM=1) .OR. SIZE(MA,DIM=2) /= SIZE(D,DIM=2)) THEN
        DO J = 1, SIZE(D,DIM=1)

```

```

        DO K = 1, SIZE(D,DIM=2)
            D(J,K) = RUNKNO
        ENDDO
    ENDDO
    RETURN
ENDIF
DO J = 1, SIZE(MA,DIM=1)
    DO K = 1, SIZE(MA,DIM=2)
        CALL FMM2DI(MA(J,K)%MFM,D(J,K))
    ENDDO
ENDDO
END SUBROUTINE FMEQ_DI2FM2

SUBROUTINE FMEQ_IM1DI(MA,D)
    USE FMVALS
    IMPLICIT NONE
    TYPE (IM), DIMENSION(:) :: MA
    INTEGER :: J,N
    INTEGER (DOUBLE_INT) :: D
    INTENT (INOUT) :: MA
    INTENT (IN) :: D
    TYPE(MULTI), SAVE :: MTLVIM
    N = SIZE(MA)
    CALL IMDI2M(D,MTLVIM)
    DO J = 1, N
        CALL IMEQ(MTLVIM,MA(J)%MIM)
    ENDDO
END SUBROUTINE FMEQ_IM1DI

SUBROUTINE FMEQ_DI1IM(D,MA)
    USE FMVALS
    IMPLICIT NONE
    TYPE (IM) :: MA
    INTEGER (DOUBLE_INT), DIMENSION(:) :: D
    INTEGER (DOUBLE_INT) :: D2
    INTEGER :: J,N
    INTENT (INOUT) :: D
    INTENT (IN) :: MA
    CALL FM_UNDEF_INP(MA)
    N = SIZE(D)
    CALL IMM2DI(MA%MIM,D2)
    DO J = 1, N
        D(J) = D2
    ENDDO
END SUBROUTINE FMEQ_DI1IM

SUBROUTINE FMEQ_IM1DI1(MA,D)
    USE FMVALS
    IMPLICIT NONE
    TYPE (IM), DIMENSION(:) :: MA
    INTEGER :: J,N
    INTEGER (DOUBLE_INT), DIMENSION(:) :: D
    INTENT (INOUT) :: MA
    INTENT (IN) :: D
    TYPE(MULTI), SAVE :: MTLVIM
    IF (SIZE(MA) /= SIZE(D)) THEN
        CALL IMST2M(' UNKNOWN ',MTLVIM)
        DO J = 1, SIZE(MA)

```

```

        CALL IMEQ(MTLVIM,MA(J)%MIM)
    ENDDO
    RETURN
ENDIF
N = SIZE(MA)
DO J = 1, N
    CALL IMDI2M(D(J),MA(J)%MIM)
ENDDO
END SUBROUTINE FMEQ_IM1DI1

```

```

SUBROUTINE FMEQ_DI1IM1(D,MA)
    USE FMVALS
    IMPLICIT NONE
    TYPE (IM), DIMENSION(:) :: MA
    INTEGER (DOUBLE_INT), DIMENSION(:) :: D
    INTEGER :: J,N
    INTENT (INOUT) :: D
    INTENT (IN) :: MA
    CALL FM_UNDEF_INP(MA)
    IF (SIZE(MA) /= SIZE(D)) THEN
        DO J = 1, SIZE(MA)
            D(J) = RUNKNO
        ENDDO
        RETURN
    ENDIF
    N = SIZE(D)
    DO J = 1, N
        CALL IMM2DI(MA(J)%MIM,D(J))
    ENDDO
END SUBROUTINE FMEQ_DI1IM1

```

```

SUBROUTINE FMEQ_IM2DI(MA,D)
    USE FMVALS
    IMPLICIT NONE
    TYPE (IM), DIMENSION(:, :) :: MA
    INTEGER :: J,K
    INTEGER (DOUBLE_INT) :: D
    INTENT (INOUT) :: MA
    INTENT (IN) :: D
    TYPE(MULTI), SAVE :: MTLVIM
    CALL IMDI2M(D,MTLVIM)
    DO J = 1, SIZE(MA,DIM=1)
        DO K = 1, SIZE(MA,DIM=2)
            CALL IMEQ(MTLVIM,MA(J,K)%MIM)
        ENDDO
    ENDDO
END SUBROUTINE FMEQ_IM2DI

```

```

SUBROUTINE FMEQ_DI2IM(D,MA)
    USE FMVALS
    IMPLICIT NONE
    TYPE (IM) :: MA
    INTEGER (DOUBLE_INT), DIMENSION(:, :) :: D
    INTEGER (DOUBLE_INT) :: D2
    INTEGER :: J,K
    INTENT (INOUT) :: D
    INTENT (IN) :: MA
    CALL FM_UNDEF_INP(MA)

```