```
 module fm_quad_int

!  FM_quadint 1.4                  David M. Smith                  Quadruple Length Integer Support

!  This module extends the definition of basic FM types (fm), (im), and (zm) so they can interact
!  with quad length integer variables.

!  Warning:  This module is needed only when the user's program explicitly declares quad length
!            integer variables.  If quad 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 quad 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.  Some compilers offered 128-bit
!  quad integers implemented in software, giving values up to 2**127 - 1, with 39 decimal digits.

!  So selected_int_kind(30) could be used to select this 128-bit format.

!  The routines in this interface extend basic functions like to_fm, to_im, to_zm so they can be
!  used with quad integer arguments.  New conversion function to_quad_int will take fm, im, or ZM
!  inputs and convert to quad 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 :: quad_int = selected_int_kind(30)

    interface to_fm
       module procedure fm_qi
       module procedure fm_qi1
       module procedure fm_qi2
    end interface

    interface to_im
       module procedure im_qi
       module procedure im_qi1
       module procedure im_qi2
    end interface

    interface to_zm
       module procedure zm_qi
       module procedure zm2_qi
       module procedure zm_qi1
       module procedure zm_qi2
    end interface

    interface to_quad_int
       module procedure fm_2qi
       module procedure im_2qi
       module procedure zm_2qi
```

```
   module procedure fm_2qi1
   module procedure im_2qi1
   module procedure zm_2qi1
   module procedure fm_2qi2
   module procedure im_2qi2
   module procedure zm_2qi2
end interface

interface assignment (=)
   module procedure fmeq_qifm
   module procedure fmeq_qiim
   module procedure fmeq_qizm
   module procedure fmeq_fmqi
   module procedure fmeq_imqi
   module procedure fmeq_zmqi
   module procedure fmeq_fm1qi
   module procedure fmeq_qi1fm
   module procedure fmeq_fm1qi1
   module procedure fmeq_qi1fm1
   module procedure fmeq_im1qi
   module procedure fmeq_qi1im
   module procedure fmeq_im1qi1
   module procedure fmeq_qi1im1
   module procedure fmeq_zm1qi
   module procedure fmeq_qi1zm
   module procedure fmeq_zm1qi1
   module procedure fmeq_qi1zm1
   module procedure fmeq_fm2qi
   module procedure fmeq_qi2fm
   module procedure fmeq_fm2qi2
   module procedure fmeq_qi2fm2
   module procedure fmeq_im2qi
   module procedure fmeq_qi2im
   module procedure fmeq_im2qi2
   module procedure fmeq_qi2im2
   module procedure fmeq_zm2qi
   module procedure fmeq_qi2zm
   module procedure fmeq_zm2qi2
   module procedure fmeq_qi2zm2
end interface

interface operator (==)
   module procedure fmleq_qifm
   module procedure fmleq_qiim
   module procedure fmleq_qizm
   module procedure fmleq_fmqi
   module procedure fmleq_imqi
   module procedure fmleq_zmqi
end interface

interface operator (/=)
   module procedure fmlne_qifm
   module procedure fmlne_qiim
   module procedure fmlne_qizm
   module procedure fmlne_fmqi
   module procedure fmlne_imqi
   module procedure fmlne_zmqi
end interface
```

```
interface operator (>)
   module procedure fmlgt_qifm
   module procedure fmlgt_qiim
   module procedure fmlgt_fmqi
   module procedure fmlgt_imqi
end interface

interface operator (>=)
   module procedure fmlge_qifm
   module procedure fmlge_qiim
   module procedure fmlge_fmqi
   module procedure fmlge_imqi
end interface

interface operator (<)
   module procedure fmllt_qifm
   module procedure fmllt_qiim
   module procedure fmllt_fmqi
   module procedure fmllt_imqi
end interface

interface operator (<=)
   module procedure fmlle_qifm
   module procedure fmlle_qiim
   module procedure fmlle_fmqi
   module procedure fmlle_imqi
end interface

interface operator (+)
   module procedure fmadd_qifm
   module procedure fmadd_qiim
   module procedure fmadd_qizm
   module procedure fmadd_fmqi
   module procedure fmadd_imqi
   module procedure fmadd_zmqi
   module procedure fmadd_qifm1
   module procedure fmadd_qiim1
   module procedure fmadd_fmqi1
   module procedure fmadd_fm1qi
   module procedure fmadd_qi1fm
   module procedure fmadd_qi1fm1
   module procedure fmadd_fm1qi1
   module procedure fmadd_imqi1
   module procedure fmadd_im1qi
   module procedure fmadd_qi1im
   module procedure fmadd_qi1im1
   module procedure fmadd_im1qi1
   module procedure fmadd_qizm1
   module procedure fmadd_zmqi1
   module procedure fmadd_zm1qi
   module procedure fmadd_qi1zm
   module procedure fmadd_qi1zm1
   module procedure fmadd_zm1qi1
   module procedure fmadd_qifm2
   module procedure fmadd_qiim2
   module procedure fmadd_fmqi2
   module procedure fmadd_fm2qi
```

```
   module procedure fmadd_qi2fm
   module procedure fmadd_qi2fm2
   module procedure fmadd_fm2qi2
   module procedure fmadd_imqi2
   module procedure fmadd_im2qi
   module procedure fmadd_qi2im
   module procedure fmadd_qi2im2
   module procedure fmadd_im2qi2
   module procedure fmadd_qizm2
   module procedure fmadd_zmqi2
   module procedure fmadd_zm2qi
   module procedure fmadd_qi2zm
   module procedure fmadd_qi2zm2
   module procedure fmadd_zm2qi2
end interface

interface operator (-)
   module procedure fmsub_qifm
   module procedure fmsub_qiim
   module procedure fmsub_qizm
   module procedure fmsub_fmqi
   module procedure fmsub_imqi
   module procedure fmsub_zmqi
   module procedure fmsub_qifm1
   module procedure fmsub_qiim1
   module procedure fmsub_fmqi1
   module procedure fmsub_fm1qi
   module procedure fmsub_qi1fm
   module procedure fmsub_qi1fm1
   module procedure fmsub_fm1qi1
   module procedure fmsub_imqi1
   module procedure fmsub_im1qi
   module procedure fmsub_qi1im
   module procedure fmsub_qi1im1
   module procedure fmsub_im1qi1
   module procedure fmsub_qizm1
   module procedure fmsub_zmqi1
   module procedure fmsub_zm1qi
   module procedure fmsub_qi1zm
   module procedure fmsub_qi1zm1
   module procedure fmsub_zm1qi1
   module procedure fmsub_qifm2
   module procedure fmsub_qiim2
   module procedure fmsub_fmqi2
   module procedure fmsub_fm2qi
   module procedure fmsub_qi2fm
   module procedure fmsub_qi2fm2
   module procedure fmsub_fm2qi2
   module procedure fmsub_imqi2
   module procedure fmsub_im2qi
   module procedure fmsub_qi2im
   module procedure fmsub_qi2im2
   module procedure fmsub_im2qi2
   module procedure fmsub_qizm2
   module procedure fmsub_zmqi2
   module procedure fmsub_zm2qi
   module procedure fmsub_qi2zm
   module procedure fmsub_qi2zm2
```

```
   module procedure fmsub_zm2qi2
end interface

interface operator (*)
   module procedure fmmpy_qifm
   module procedure fmmpy_qiim
   module procedure fmmpy_qizm
   module procedure fmmpy_fmqi
   module procedure fmmpy_imqi
   module procedure fmmpy_zmqi
   module procedure fmmpy_qifm1
   module procedure fmmpy_qiim1
   module procedure fmmpy_fmqi1
   module procedure fmmpy_fm1qi
   module procedure fmmpy_qi1fm
   module procedure fmmpy_qi1fm1
   module procedure fmmpy_fm1qi1
   module procedure fmmpy_imqi1
   module procedure fmmpy_im1qi
   module procedure fmmpy_qi1im
   module procedure fmmpy_qi1im1
   module procedure fmmpy_im1qi1
   module procedure fmmpy_qizm1
   module procedure fmmpy_zmqi1
   module procedure fmmpy_zm1qi
   module procedure fmmpy_qi1zm
   module procedure fmmpy_qi1zm1
   module procedure fmmpy_zm1qi1
   module procedure fmmpy_qifm2
   module procedure fmmpy_qiim2
   module procedure fmmpy_fmqi2
   module procedure fmmpy_fm2qi
   module procedure fmmpy_qi2fm
   module procedure fmmpy_qi2fm2
   module procedure fmmpy_fm2qi2
   module procedure fmmpy_imqi2
   module procedure fmmpy_im2qi
   module procedure fmmpy_qi2im
   module procedure fmmpy_qi2im2
   module procedure fmmpy_im2qi2
   module procedure fmmpy_qizm2
   module procedure fmmpy_zmqi2
   module procedure fmmpy_zm2qi
   module procedure fmmpy_qi2zm
   module procedure fmmpy_qi2zm2
   module procedure fmmpy_zm2qi2
end interface

interface operator (/)
   module procedure fmdiv_qifm
   module procedure fmdiv_qiim
   module procedure fmdiv_qizm
   module procedure fmdiv_fmqi
   module procedure fmdiv_imqi
   module procedure fmdiv_zmqi
   module procedure fmdiv_qifm1
   module procedure fmdiv_qiim1
   module procedure fmdiv_fmqi1
```

```
      module procedure fmdiv_fm1qi
      module procedure fmdiv_qi1fm
      module procedure fmdiv_qi1fm1
      module procedure fmdiv_fm1qi1
      module procedure fmdiv_imqi1
      module procedure fmdiv_im1qi
      module procedure fmdiv_qi1im
      module procedure fmdiv_qi1im1
      module procedure fmdiv_im1qi1
      module procedure fmdiv_qizm1
      module procedure fmdiv_zmqi1
      module procedure fmdiv_zm1qi
      module procedure fmdiv_qi1zm
      module procedure fmdiv_qi1zm1
      module procedure fmdiv_zm1qi1
      module procedure fmdiv_qifm2
      module procedure fmdiv_qiim2
      module procedure fmdiv_fmqi2
      module procedure fmdiv_fm2qi
      module procedure fmdiv_qi2fm
      module procedure fmdiv_qi2fm2
      module procedure fmdiv_fm2qi2
      module procedure fmdiv_imqi2
      module procedure fmdiv_im2qi
      module procedure fmdiv_qi2im
      module procedure fmdiv_qi2im2
      module procedure fmdiv_im2qi2
      module procedure fmdiv_qizm2
      module procedure fmdiv_zmqi2
      module procedure fmdiv_zm2qi
      module procedure fmdiv_qi2zm
      module procedure fmdiv_qi2zm2
      module procedure fmdiv_zm2qi2
   end interface

   interface operator (**)
      module procedure fmpwr_qifm
      module procedure fmpwr_qiim
      module procedure fmpwr_qizm
      module procedure fmpwr_fmqi
      module procedure fmpwr_imqi
      module procedure fmpwr_zmqi
   end interface

 contains

      subroutine fmqi2m(ival, ma)

!  Convert quad length integer ival to multiple precision ma.

      use fmvals
      implicit none

      type(multi) :: ma
      integer (quad_int) :: ival

      real (kind(1.0d0)) :: mk, ml, mval
      integer (quad_int) :: j, jm2, kb, kb1, n1, nmval, nv2
```

```
      integer :: kl
      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)
          return
      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)
              return
          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
          return
      endif

!             Compute and store the digits, right to left.

      ma%mp(2) = 0
      j = ndig + 1
```

```
      kl = 1
      do while (kl == 1)
         kl = 0
         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) then
                 kl = 1
                 cycle
             endif

!             Here ival cannot be expressed exactly.

             write (str, "(i50)") ival
             call fmst2m(str, ma)
             return
         endif
      enddo

!             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

      return
      end subroutine fmqi2m


      subroutine fmm2qi(ma, ival)

!  Convert multiple precision ma to quad length integer ival.

      use fmvals
      implicit none

      type(multi) :: ma
      integer (quad_int) :: ival

      integer (quad_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 fmm2qi


      subroutine imqi2m(ival, ma)

!  ma = ival
```

```
!  Convert a quad length integer to an IM number.

      use fmvals
      implicit none

      type(multi) :: ma
      integer (quad_int) :: ival

      integer :: ndsave
      intent (in) :: ival
      intent (inout) :: ma
      type(multi), save :: mvlvfm

      call fmqi2m(ival, mvlvfm)

      if (int(mvlvfm%mp(2)) > ndig) then
          ndsave = ndig
          ndig = max(3, int(mvlvfm%mp(2)))
          call fmqi2m(ival, mvlvfm)
          call imfm2i(mvlvfm, ma)
          ndig = ndsave
      else
          call imfm2i(mvlvfm, ma)
      endif

      return
      end subroutine imqi2m


      subroutine imm2qi(ma, ival)

!  ival = ma

!  Convert an IM number to quad length integer.

      use fmvals
      implicit none

      type(multi) :: ma
      integer (quad_int) :: ival

      integer :: ndsave
      intent (in) :: ma
      intent (inout) :: ival
      type(multi), save :: mtlvfm

      ndsave = ndig
      ndig = max(3, int(ma%mp(2)))

      call imi2fm(ma, mtlvfm)
      call fmm2qi(mtlvfm, ival)

      ndig = ndsave
      return
      end subroutine imm2qi
```

```
function fm_qi(d)     result (return_value)
   use fmvals
   implicit none
   type (fm) :: return_value
   integer (quad_int) :: d
   intent (in) :: d
   call fmqi2m(d, return_value%mfm)
end function fm_qi

function fm_qi1(d)     result (return_value)
   use fmvals
   implicit none
   integer (quad_int), dimension(:) :: d
   type (fm), dimension(size(d)) :: return_value
   integer :: j, n
   intent (in) :: d
   n = size(d)
   do j = 1, n
      call fmqi2m(d(j), return_value(j)%mfm)
   enddo
end function fm_qi1

function fm_qi2(d)     result (return_value)
   use fmvals
   implicit none
   integer (quad_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 fmqi2m(d(j, k), return_value(j, k)%mfm)
      enddo
   enddo
end function fm_qi2

function im_qi(d)     result (return_value)
   use fmvals
   implicit none
   type (im) :: return_value
   integer (quad_int) :: d
   intent (in) :: d
   call imqi2m(d, return_value%mim)
end function im_qi

function im_qi1(d)     result (return_value)
   use fmvals
   implicit none
   integer (quad_int), dimension(:) :: d
   type (im), dimension(size(d)) :: return_value
   integer :: j, n
   intent (in) :: d
   n = size(d)
   do j = 1, n
      call imqi2m(d(j), return_value(j)%mim)
   enddo
end function im_qi1
```

```
function im_qi2(d)     result (return_value)
   use fmvals
   implicit none
   integer (quad_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 imqi2m(d(j, k), return_value(j, k)%mim)
      enddo
   enddo
end function im_qi2

function zm_qi(d)     result (return_value)
   use fmvals
   implicit none
   type (zm) :: return_value
   integer (quad_int) :: d
   intent (in) :: d
   type(multi), save :: mtlvfm, mulvfm
   call fmqi2m(d, mtlvfm)
   call fmi2m(0, mulvfm)
   call zmcmpx(mtlvfm, mulvfm, return_value%mzm)
end function zm_qi

function zm2_qi(d1, d2)     result (return_value)
   use fmvals
   implicit none
   type (zm) :: return_value
   integer (quad_int) :: d1, d2
   intent (in) :: d1, d2
   type(multi), save :: mtlvfm, mulvfm
   call fmqi2m(d1, mtlvfm)
   call fmqi2m(d2, mulvfm)
   call zmcmpx(mtlvfm, mulvfm, return_value%mzm)
end function zm2_qi

function zm_qi1(d)     result (return_value)
   use fmvals
   implicit none
   integer (quad_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 fmqi2m(d(j), mtlvfm)
      call zmcmpx(mtlvfm, mulvfm, return_value(j)%mzm)
   enddo
end function zm_qi1

function zm_qi2(d)     result (return_value)
   use fmvals
   implicit none
   integer (quad_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 fmqi2m(d(j, k), mtlvfm)
         call zmcmpx(mtlvfm, mulvfm, return_value(j, k)%mzm)
      enddo
   enddo
end function zm_qi2

function fm_2qi(ma)     result (return_value)
   use fmvals
   implicit none
   type (fm) :: ma
   integer (quad_int) :: return_value
   intent (in) :: ma
   call fm_undef_inp(ma)
   call fmm2qi(ma%mfm, return_value)
end function fm_2qi

function im_2qi(ma)     result (return_value)
   use fmvals
   implicit none
   type (im) :: ma
   integer (quad_int) :: return_value
   intent (in) :: ma
   call fm_undef_inp(ma)
   call imm2qi(ma%mim, return_value)
end function im_2qi

function zm_2qi(ma)     result (return_value)
   use fmvals
   implicit none
   type (zm) :: ma
   integer (quad_int) :: return_value
   intent (in) :: ma
   type(multi), save :: mtlvfm
   call fm_undef_inp(ma)
   call zmreal(ma%mzm, mtlvfm)
   call fmm2qi(mtlvfm, return_value)
end function zm_2qi

function fm_2qi1(ma)     result (return_value)
   use fmvals
   implicit none
   type (fm), dimension(:) :: ma
   integer (quad_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 fmm2qi(ma(j)%mfm, return_value(j))
   enddo
end function fm_2qi1
```

```
function im_2qi1(ma)     result (return_value)
   use fmvals
   implicit none
   type (im), dimension(:) :: ma
   integer (quad_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 imm2qi(ma(j)%mim, return_value(j))
   enddo
end function im_2qi1

function zm_2qi1(ma)     result (return_value)
   use fmvals
   implicit none
   type (zm), dimension(:) :: ma
   integer (quad_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 fmm2qi(ma(j)%mzm(1), return_value(j))
   enddo
end function zm_2qi1

function fm_2qi2(ma)     result (return_value)
   use fmvals
   implicit none
   type (fm), dimension(:,:) :: ma
   integer (quad_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 fmm2qi(ma(j, k)%mfm, return_value(j, k))
      enddo
   enddo
end function fm_2qi2

function im_2qi2(ma)     result (return_value)
   use fmvals
   implicit none
   type (im), dimension(:,:) :: ma
   integer (quad_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 imm2qi(ma(j, k)%mim, return_value(j, k))
      enddo
   enddo
end function im_2qi2
```

```
function zm_2qi2(ma)     result (return_value)
   use fmvals
   implicit none
   type (zm), dimension(:,:) :: ma
   integer (quad_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 fmm2qi(ma(j, k)%mzm(1), return_value(j, k))
      enddo
   enddo
end function zm_2qi2


subroutine fmeq_qifm(d, ma)
   use fmvals
   implicit none
   type (fm) :: ma
   integer (quad_int) :: d
   intent (inout) :: d
   intent (in) :: ma
   call fm_undef_inp(ma)
   call fmm2qi(ma%mfm, d)
end subroutine fmeq_qifm

subroutine fmeq_qiim(d, ma)
   use fmvals
   implicit none
   type (im) :: ma
   integer (quad_int) :: d
   intent (inout) :: d
   intent (in) :: ma
   call fm_undef_inp(ma)
   call imm2qi(ma%mim, d)
end subroutine fmeq_qiim

subroutine fmeq_qizm(d, ma)
   use fmvals
   implicit none
   type (zm) :: ma
   integer (quad_int) :: d
   intent (inout) :: d
   intent (in) :: ma
   type(multi), save :: mtlvfm
   call fm_undef_inp(ma)
   call zmreal(ma%mzm, mtlvfm)
   call fmm2qi(mtlvfm, d)
end subroutine fmeq_qizm

subroutine fmeq_fmqi(ma, d)
   use fmvals
   implicit none
   type (fm) :: ma
   integer (quad_int) :: d
   intent (inout) :: ma
```

```
   intent (in) :: d
   call fmqi2m(d, ma%mfm)
end subroutine fmeq_fmqi

subroutine fmeq_imqi(ma, d)
   use fmvals
   implicit none
   type (im) :: ma
   integer :: ival
   integer (quad_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_imqi

subroutine fmeq_zmqi(ma, d)
   use fmvals
   implicit none
   type (zm) :: ma
   integer (quad_int) :: d
   intent (inout) :: ma
   intent (in) :: d
   type(multi), save :: mtlvfm, mulvfm
   call fmqi2m(d, mtlvfm)
   call fmi2m(0, mulvfm)
   call zmcmpx(mtlvfm, mulvfm, ma%mzm)
end subroutine fmeq_zmqi

subroutine fmeq_fm1qi(ma, d)
   use fmvals
   implicit none
   type (fm), dimension(:) :: ma
   integer :: j, n
   integer (quad_int) :: d
   intent (inout) :: ma
   intent (in) :: d
   type(multi), save :: mtlvfm
   n = size(ma)
   call fmqi2m(d, mtlvfm)
   do j = 1, n
      call fmeq(mtlvfm, ma(j)%mfm)
   enddo
end subroutine fmeq_fm1qi

subroutine fmeq_qi1fm(d, ma)
   use fmvals
   implicit none
   type (fm) :: ma
   integer (quad_int), dimension(:) :: d
   integer (quad_int) :: d2
   integer :: j, n
```

```
   intent (inout) :: d
   intent (in) :: ma
   call fm_undef_inp(ma)
   n = size(d)
   call fmm2qi(ma%mfm, d2)
   do j = 1, n
      d(j) = d2
   enddo
end subroutine fmeq_qi1fm

subroutine fmeq_fm1qi1(ma, d)
   use fmvals
   implicit none
   type (fm), dimension(:) :: ma
   integer :: j, n
   integer (quad_int), dimension(:) :: d
   intent (inout) :: ma
   intent (in) :: d
   type(multi), save :: mtlvfm
   if (size(ma) /= size(d)) then
       call fmunknown(mtlvfm)
       do j = 1, size(ma)
          call fmeq(mtlvfm, ma(j)%mfm)
       enddo
       return
   endif
   n = size(ma)
   do j = 1, n
      call fmqi2m(d(j), ma(j)%mfm)
   enddo
end subroutine fmeq_fm1qi1

subroutine fmeq_qi1fm1(d, ma)
   use fmvals
   implicit none
   type (fm), dimension(:) :: ma
   integer (quad_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 fmm2qi(ma(j)%mfm, d(j))
   enddo
end subroutine fmeq_qi1fm1

subroutine fmeq_fm2qi(ma, d)
   use fmvals
   implicit none
   type (fm), dimension(:,:) :: ma
   integer :: j, k
```

```
   integer (quad_int) :: d
   intent (inout) :: ma
   intent (in) :: d
   type(multi), save :: mtlvfm
   call fmqi2m(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_fm2qi

subroutine fmeq_qi2fm(d, ma)
   use fmvals
   implicit none
   type (fm) :: ma
   integer (quad_int), dimension(:,:) :: d
   integer (quad_int) :: d2
   integer :: j, k
   intent (inout) :: d
   intent (in) :: ma
   call fm_undef_inp(ma)
   call fmm2qi(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_qi2fm

subroutine fmeq_fm2qi2(ma, d)
   use fmvals
   implicit none
   type (fm), dimension(:,:) :: ma
   integer :: j, k
   integer (quad_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 fmunknown(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 fmqi2m(d(j, k), ma(j, k)%mfm)
      enddo
   enddo
end subroutine fmeq_fm2qi2

subroutine fmeq_qi2fm2(d, ma)
   use fmvals
   implicit none
```

```
   type (fm), dimension(:,:) :: ma
   integer (quad_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 fmm2qi(ma(j, k)%mfm, d(j, k))
      enddo
   enddo
end subroutine fmeq_qi2fm2

subroutine fmeq_im1qi(ma, d)
   use fmvals
   implicit none
   type (im), dimension(:) :: ma
   integer :: j, n
   integer (quad_int) :: d
   intent (inout) :: ma
   intent (in) :: d
   type(multi), save :: mtlvim
   n = size(ma)
   call imqi2m(d, mtlvim)
   do j = 1, n
      call imeq(mtlvim, ma(j)%mim)
   enddo
end subroutine fmeq_im1qi

subroutine fmeq_qi1im(d, ma)
   use fmvals
   implicit none
   type (im) :: ma
   integer (quad_int), dimension(:) :: d
   integer (quad_int) :: d2
   integer :: j, n
   intent (inout) :: d
   intent (in) :: ma
   call fm_undef_inp(ma)
   n = size(d)
   call imm2qi(ma%mim, d2)
   do j = 1, n
      d(j) = d2
   enddo
end subroutine fmeq_qi1im

subroutine fmeq_im1qi1(ma, d)
   use fmvals
   implicit none
   type (im), dimension(:) :: ma
```

```
   integer :: j, n
   integer (quad_int), dimension(:) :: d
   intent (inout) :: ma
   intent (in) :: d
   type(multi), save :: mtlvim
   if (size(ma) /= size(d)) then
       call imunknown(mtlvim)
       do j = 1, size(ma)
          call imeq(mtlvim, ma(j)%mim)
       enddo
       return
   endif
   n = size(ma)
   do j = 1, n
      call imqi2m(d(j), ma(j)%mim)
   enddo
end subroutine fmeq_im1qi1

subroutine fmeq_qi1im1(d, ma)
   use fmvals
   implicit none
   type (im), dimension(:) :: ma
   integer (quad_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 imm2qi(ma(j)%mim, d(j))
   enddo
end subroutine fmeq_qi1im1

subroutine fmeq_im2qi(ma, d)
   use fmvals
   implicit none
   type (im), dimension(:,:) :: ma
   integer :: j, k
   integer (quad_int) :: d
   intent (inout) :: ma
   intent (in) :: d
   type(multi), save :: mtlvim
   call imqi2m(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_im2qi

subroutine fmeq_qi2im(d, ma)
   use fmvals
```