

```
BBBBBBBBBBBBBB      AAAAAAAAAA      SSSSSSSSSSSS      RRRRRRRRRRRR      TTTTTTTTTTTTTT      LLL
BBBBBBBBBBBBBB      AAAAAAAAAA      SSSSSSSSSSSS      RRRRRRRRRRRR      TTTTTTTTTTTTTT      LLL
BBBBBBBBBBBBBB      AAAAAAAAAA      SSSSSSSSSSSS      RRRRRRRRRRRR      TTTTTTTTTTTTTT      LLL
BBB      BBB      AAA      AAA      SSS      RRR      RRR      TTT      LLL
BBB      BBB      AAA      AAA      SSS      RRR      RRR      TTT      LLL
BBB      BBB      AAA      AAA      SSS      RRR      RRR      TTT      LLL
BBB      BBB      AAA      AAA      SSS      RRR      RRR      TTT      LLL
BBB      BBB      AAA      AAA      SSS      RRR      RRR      TTT      LLL
BBB      BBB      AAA      AAA      SSS      RRR      RRR      TTT      LLL
BBBBBBBBBBBBBB      AAA      AAA      SSS      RRR      RRR      TTT      LLL
BBBBBBBBBBBBBB      AAA      AAA      SSS      RRR      RRR      TTT      LLL
BBBBBBBBBBBBBB      AAA      AAA      SSS      RRR      RRR      TTT      LLL
BBB      BBB      AAAAAAAAAAAAAAAAAA      SSS      RRR      RRR      TTT      LLL
BBB      BBB      AAAAAAAAAAAAAAAAAA      SSS      RRR      RRR      TTT      LLL
BBB      BBB      AAAAAAAAAAAAAAAAAA      SSS      RRR      RRR      TTT      LLL
BBB      BBB      AAA      AAA      SSS      RRR      RRR      TTT      LLL
BBB      BBB      AAA      AAA      SSS      RRR      RRR      TTT      LLL
BBB      BBB      AAA      AAA      SSS      RRR      RRR      TTT      LLL
BBBBBBBBBBBBBB      AAA      AAA      SSSSSSSSSSSS      RRR      RRR      TTT      LLLLLLLLLLLLLLLL
BBBBBBBBBBBBBB      AAA      AAA      SSSSSSSSSSSS      RRR      RRR      TTT      LLLLLLLLLLLLLLLL
BBBBBBB BBB      AAA      AAA      SSSSSSSSSSSS      RRR      RRR      TTT      LLLLLLLLLLLLLLLL
```

```
MM      MM      AAAAAA      TTTTTTTTTT      RRRRRRRR      IIIII'I      XX      XX
MM      MM      AAAAAA      TTTTTTTTTT      RRRRRRRR      IIIIII      XX      XX
MMMM    MMMM    AA      AA      TT      RR      RR      II      XX      XX
MMMM    MMMM    AA      AA      TT      RR      RR      II      XX      XX
MM      MM      AA      AA      TT      RR      RR      II      XX      XX
MM      MM      AA      AA      TT      RR      RR      II      XX      XX
MM      MM      AA      AA      TT      RRRRRRRR      II      XX      XX
MM      MM      AA      AA      TT      RRRRRRRR      II      XX      XX
MM      MM      AAAAAAAAAA      TT      RR      RR      II      XX      XX
MM      MM      AAAAAAAAAA      TT      RR      RR      II      XX      XX
MM      MM      AA      AA      TT      RR      RR      II      XX      XX
MM      MM      AA      AA      TT      RR      RR      II      XX      XX
MM      MM      AA      AA      TT      RR      RR      IIIIII      XX      XX
MM      MM      AA      AA      TT      RR      RR      IIIIII      XX      XX
```

```
MM      MM      AAAAAA      RRRRRRRR
MM      MM      AAAAAA      RRRRRRRR
MMMM    MMMM    AA      AA      RR      RR
MMMM    MMMM    AA      AA      RR      RR
MM      MM      AA      AA      RR      RR
MM      MM      AA      AA      RR      RR
MM      MM      AA      AA      RRRRRRRR
MM      MM      AA      AA      RRRRRRRR
MM      MM      AAAAAAAAAA      RR      RR
MM      MM      AAAAAAAAAA      RR      RR
MM      MM      AA      AA      RR      RR
MM      MM      AA      AA      RR      RR
MM      MM      AA      AA      RR      RR
MM      MM      AA      AA      RR      RR
```

.TITLE MATRIX
.IDENT /1-004/

; File: MATRIX.MAR Edit: LEB1004

```

*****
*
* COPYRIGHT (c) 1978, 1980, 1982, 1984 BY
* DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.
* ALL RIGHTS RESERVED.
*
* THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED
* ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE AND WITH THE
* INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR ANY OTHER
* COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY
* OTHER PERSON. NO TITLE TO AND OWNERSHIP OF THE SOFTWARE IS HEREBY
* TRANSFERRED.
*
* THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT NOTICE
* AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT
* CORPORATION.
*
* DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF ITS
* SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.
*
*****

```

++
: FACILITY: BASIC code support

: ABSTRACT:

: ENVIRONMENT: User Mode, AST Reentrant

--
: MODIFIED BY:

- : 1-001 - Original - contains FETCH and STORE macros
- : 1-002 - Added code to handle arrays of descriptors. Needed to add
label declarations at beginning of each macro. LEB 23-June-1982
- : 1-003 - In the FETCH macro, after calling BASS\$FETCH_BFA the value must
be moved into R0. PLL 28-Jun-1982
- : 1-004 - Use offsets from the stack. LEB 9-Jul-1982

.SBTTL DECLARATIONS

INCLUDE FILES:

```

$DSCDEF      ; define descriptor offsets
$SFDEF       ; use to get scale

```

EXTERNAL DECLARATIONS:

```

.DSABL  GBL      ; Prevent undeclared
                ; symbols from being
                ; automatically global.
.EXTRN  BASSK_ARGDON AT ; signalled if all 3 blocks
                ; not present in array desc
                ; or dimct = 0
.EXTRN  BASSK_DATTYPERR ; signalled if dtype of array
                ; isn't word long float double
.EXTRN  BASSK_MATDIMERR ; signalled if # of dims on
                ; source arrays don't agree
.EXTRN  BASSK_ARRMUSSAM ; signalled if upper and lower
                ; bnds not same on src arrays
.EXTRN  BASSSTO_FA_W_R8 ; array element store for word
.EXTRN  BASSSTO_FA_L_R8 ; array element store for long
.EXTRN  BASSSTO_FA_F_R8 ; array element store - float
.EXTRN  BASSSTO_FA_D_R8 ; array element store - double
.EXTRN  BASSSTO_FA_B_R8 ; array element store - byte
.EXTRN  BASSSTO_FA_G_R8 ; array element store - gfloat
.EXTRN  BASSSTO_FA_H_R8 ; array element store - hfloat
.EXTRN  BASSFET_FA_W_R8 ; array element fetch - word
.EXTRN  BASSFET_FA_L_R8 ; array element fetch - long
.EXTRN  BASSFET_FA_F_R8 ; array element fetch - float
.EXTRN  BASSFET_FA_D_R8 ; array element fetch - double
.EXTRN  BASSFET_FA_B_R8 ; array element fetch - byte
.EXTRN  BASSFET_FA_G_R8 ; array element fetch - gfloat
.EXTRN  BASSFET_FA_H_R8 ; array element fetch - hfloat
.EXTRN  BASSMAT_REDIM   ; check if redimensioning of
                ; dest array is necessary, if
                ; so, do it
.EXTRN  BASS$SCALE_R1  ; scale for double precision
.EXTRN  MTH$DINT_RZ    ; truncate dbl precision number
.EXTRN  BASS$STOP      ; signal fatal errors
.EXTRN  BASSFETCH_BFA
.EXTRN  BASSSTORE_BFA

```

MACROS:

```

FETCH      fetch an element from an array
STORE      store an element into an array

```

EQUATED SYMBOLS:

```

: lower_bnd2 = 0           : stack offset for temp
: lower_bnd1 = 4           : stack offset for temp
: upper_bnd1 = 8           : stack offset for temp
: save_src1 = 12          : stack offset for temp
: dsc$l_l1_1 = 24          : desc offset if 1 sub
: dsc$l_u1_1 = 28          : desc offset if 1 sub
: dsc$l_l1_2 = 28          : desc offset if 2 sub
: dsc$l_u1_2 = 32          : desc offset if 2 sub
: dsc$l_l2_2 = 36          : desc offset if 2 sub
: dsc$l_u2_2 = 40          : desc offset if 2 sub

```

```

: OWN STORAGE:
:

```

```

: PSECT DECLARATIONS:
:

```

```

.PSECT _BAS$CODE PIC,USR,CON,REL,LCL,SHR,-
      EXE, RD, NOWRT, LONG

```

```

:
: These macros are a substitute for calls to the array fetch and store
: routines. They will call the BASS routines only if the array is a
: virtual array. Otherwise, they will calculate the linear index into
: the array via the INDEX instruction. (Note that BASIC programs must
: be able to handle FORTRAN arrays, so the code must check for arrays
: stored by column.) The INDEX instructions should provide a significant
: performance improvement over calling a routine for each element of
: the array.
:
:

```

```

.MACRO FETCH array_dtype,?L1,?L2,?L3,?L10,?L11,?L12,?L21 ; fetch an array element

CMPB dsc$b_dtype(R0), #dsc$k_dtype_desc ; descriptor?
BNEQ L10
MOVL 4(R0), R4 ; fetch addr of descriptor
MOVB dsc$b_dtype(R4), dtype(SP) ; store data type from desc
MOVB dsc$b_class(R4), class(SP) ; store class from desc
MOVAL data(SP), pointer(SP)
MOVW #10, str_len(SP)
CMPB dsc$b_dimct(R0), #1 ; check # of dimensions
BNEQ L12 ; branch if 2 dimensions
PUSHL R1 ; value of 1st index
PUSHAL value_desc+4(SP) ; addr of value desc
PUSHL R0 ; addr of array desc
CALLS #3,G^BASSFETCH_BFA
MOV'array_dtype' data(SP), R0 ; put value into R0
BRW L3
L12: PUSHL R2 ; value of 2nd index
PUSHL R1 ; value of 1st index
PUSHAL value_desc+8(SP) ; addr of value desc
PUSHL R0 ; addr of array desc
CALLS #4,G^BASSFETCH_BFA
MOV'array_dtype' data(SP), R0 ; put value into R0
BRW L3
L10: CMPB dsc$b_class(R0), #dsc$k_class_bfa ;virtual array?
BNEQ L1 ; no
JSB G^BASSFET_FA_'array_dtype'_R8 ; yes, use the fetch routine
BRW L3 ; done
L1: BBS #5, 10(R0), L2 ; br if array stored by cols
CMPB dsc$b_dimct(R0), #1 ; 1 or 2 dims?
BNEQ L11 ; 2 dims
MOVZWL dsc$w_length(R0), R4 ; make length longword
INDEX R1, dsc$l_l1_1(R0), dsc$l_u1_1(R0), R4, #0, R3
ADDL dsc$a_a0(R0), R3 ; add start addr to offset
MOV'array_dtype' (R3), R0 ; return element in R0
BRW L3 ; 1 dim done
L11: INDEX R1, dsc$l_l1_2(R0), dsc$l_u1_2(R0), dsc$l_m2(R0), #0, R3
; I * M2 [Li's are zero]
MOVZWL dsc$w_length(R0), R4 ; need longword length for INDEX
INDEX R2, dsc$l_l2_2(R0), dsc$l_u2_2(R0), R4, R3, R3
; (J + (I * M2)) * length
ADDL dsc$a_a0(R0), R3 ; compute addr of element
MOV'array_dtype' (R3), R0 ; return element in R0
BRW L3 ; done
L2: CMPB dsc$b_dimct(R0), #1 ; 1 or 2 dims?

```

```

BNEQ L21 : 2 dims
MOVZWL dsc$w_length(R0), R4 : make length longword
INDEX R2, dsc$l_l1_1(R0), dsc$l_u1_1(R0), R4, #0, R3
ADDL dsc$a_a0(R0), R3 : add start addr to offset
MOV'array_dtype' (R3), R0 : return element in R0
BRW L3 : 1 dim done
L21: INDEX R2, dsc$l_l2_2(R0), dsc$l_u2_2(R0), dsc$l_m1(R0), #0, R3
: J * M1 [Li's are zero]
MOVZWL dsc$w_length(R0), R4 : need longword length for INDEX
INDEX R1, dsc$l_l1_2(R0), dsc$l_u1_2(R0), R4, R3, R3
: (I + (J * M1)) * length
ADDL dsc$a_a0(R0), R3 : compute addr of element
MOV'array_dtype' (R3), R0 : return element in R0

L3:
.ENDM

```

```

.MACRO STORE array_dtype,?L1,?L2,?L3,?L4,?L5,?L6,?L7,?L8,?L9,?L10,?L11,?L12,?L21,?L30,?L31,?L32,?L41,?L50,?L51,?L52,?L61,
  .IF IDN 'array_dtype' H ; array is hfloat
  CMPB dsc$b_dtype(R4), #dsc$k_dtype_desc ; descriptor?
  BNEQ L10
  MOVL 4(R4), R0 ; fetch addr of descriptor
  MOVBL dsc$b_dtype(R0), dtype(SP) ; load in data type
  MOVBL dsc$b_class(R0), class(SP) ; load in class field
  MOVAL data(SP), pointer (SP)
  MOVW #10, str_len(SP)
  CMPB dsc$b_dimct(R4), #1 ; check # of dimensions
  BNEQ L12 ; branch if 2 dimensions
  PUSHL R5 ; value of 1st index
  PUSHL R4 ; addr of array desc
  PUSHAL value_desc+8(SP) ; addr of value desc
  CALLS #3,G^BAS$STORE_BFA
  BRW L9
L12:  PUSHL R6 ; value of 2nd index
      PUSHL R5 ; value of 1st index
      PUSHL R4 ; addr of array desc
      PUSHAL value_desc+12(SP) ; addr of value desc
      CALLS #4,G^BAS$STORE_BFA
      BRW L9
L10:  CMPB dsc$b_class(R4), #dsc$k_class_bfa ; virtual array?
      BNEQ L1 ; no
      JSB G^BAS$STO_FA_'array_dtype'_R8 ; yes, call store routine
      BRW L9 ; done
L1:   BBS #5, 10(R4), L2 ; br if stored row-wise
      CMPB dsc$b_dimct(R4), #1 ; 1 or 2 dim?
      BNEQ L11 ; 2 dims
      MOVZWL dsc$w_length(R4), R8 ; make length longword
      INDEX R5, dsc$l_l1_1(R4), dsc$l_u1_1(R4), R8, #0, R7
      ADDL dsc$a_a0(R4), R7 ; add start addr to offset
      MOV'array_dtype' R0, (R7) ; store element from R0
      BRW L9 ; 1 dim done
L11:  INDEX R5, dsc$l_l1_2(R4), dsc$l_u1_2(R4), dsc$l_m2(R4), #0, R7
      ; I : M2
      MOVZWL dsc$w_length(R4), R8 ; longword length for INDEX
      INDEX R6, dsc$l_l2_2(R4), dsc$l_u2_2(R4), R8, R7, R7
      ; (J + (I * M2)) * length
      ADDL dsc$a_a0(R4), R7 ; compute addr of element
      MOV'array_dtype' R0, (R7) ; store element from R0
      BRW L9 ; 1 dim done
L2:   CMPB dsc$b_dimct(R4), #1 ; 1 or 2 dim?
      BNEQ L21 ; 2 dims
      MOVZWL dsc$w_length(R4), R8 ; make length longword
      INDEX R6, dsc$l_l1_1(R4), dsc$l_u1_1(R4), R8, #0, R7
      ADDL dsc$a_a0(R4), R7 ; add start addr to offset
      MOV'array_dtype' R0, (R7) ; store element from R0
      BRW L9 ; 1 dim done
L21:  INDEX R6, dsc$l_l2_2(R4), dsc$l_u2_2(R4), dsc$l_m1(R4), #0, R7
      ; J * M1
      MOVZWL dsc$w_length(R4), R8 ; longword length for INDEX
      INDEX R5, dsc$l_l1_2(R4), dsc$l_u1_2(R4), R8, R7, R7
      ; (I + (J * M1)) * length
      ADDL dsc$a_a0(R4), R7 ; compute addr of element

```



```

MOV 'array_dtype'      R0, (R7)      ; store element from R0
.IFF
.IF      IDN      'array_dtype', G    ; array is gfloat
CMPB    dsc$b_dtype(R2), #dsc$k_dtype_dsc ; descriptor?
BNEQ    L30
MOVL    4(R2), R0                    ; fetch addr of descriptor
MOVB    dsc$b_dtype(R0), dtype(SP)    ; load in data type
MOVB    dsc$b_class(R0), class(SP)    ; load in class field
MOVAL   data(SP), pointer (SP)
MOVW    #10, str_len(SP)
CMPB    dsc$b_dimct(R2), #1          ; check # of dimensions
BNEQ    L32                          ; branch if 2 dimensions
PUSHL   R3                          ; value of 1st index
PUSHL   R2                          ; addr of array desc
PUSHAL  value_desc+8(SP)             ; addr of value desc
CALLS   #3, G^BAS$STORE_BFA
BRW     L9
L32:    PUSHL   R4                    ; value of 2nd index
PUSHL   R3                          ; value of 1st index
PUSHL   R2                          ; addr of array desc
PUSHAL  value_desc+12(SP)           ; addr of value desc
CALLS   #4, G^BAS$STORE_BFA
BRW     L9
L30:    CMPB    dsc$b_class(R2), #dsc$k_class_bfa ; virtual array?
BNEQ    L3                            ; no
JSB     G^BAS$STO_FA_'array_dtype'_R8 ; yes, call store routine
BRW     L9                            ; done
L3:     BBS     #5, 10(R2), L4         ; br if stored row-wise
CMPB    dsc$b_dimct(R2), #1          ; 1 or 2 dim?
BNEQ    L31                          ; 2 dims
MOVZWL  dsc$w_length(R2), R6         ; make length longword
INDEX   R3, dsc$l_l1_1(R2), dsc$l_u1_1(R2), R6, #0, R5
ADDL    dsc$a_a0(R2), R5             ; add start addr to offset
MOV 'array_dtype'      R0, (R5)      ; store element from R0
BRW     L9                            ; 1 dim done
L31:    INDEX   R3, dsc$l_l1_2(R2), dsc$l_u1_2(R2), dsc$l_m2(R2), #0, R5
; I * M2
MOVZWL  dsc$w_length(R2), R6         ; longword length for INDEX
INDEX   R4, dsc$l_l2_2(R2), dsc$l_u2_2(R2), R6, R5, R5
; (J + (I * M2)) * length
ADDL    dsc$a_a0(R2), R5             ; compute addr of element
MOV 'array_dtype'      R0, (R5)      ; store element from R0
BRW     L9
L4:     CMPB    dsc$b_dimct(R2), #1   ; 1 or 2 dim?
BNEQ    L41                          ; 2 dims
MOVZWL  dsc$w_length(R2), R6         ; make length longword
INDEX   R4, dsc$l_l1_1(R2), dsc$l_u1_1(R2), R6, #0, R5
ADDL    dsc$a_a0(R2), R5             ; add start addr to offset
MOV 'array_dtype'      R0, (R5)      ; store element from R0
BRW     L9                            ; 1 dim done
L41:    INDEX   R4, dsc$l_l2_2(R2), dsc$l_u2_2(R2), dsc$l_m1(R2), #0, R5
; J * M1
MOVZWL  dsc$w_length(R2), R6         ; longword length for INDEX
INDEX   R3, dsc$l_l1_2(R2), dsc$l_u1_2(R2), R6, R5, R5
; (I + (J * M1)) * length
ADDL    dsc$a_a0(R2), R5             ; compute addr of element

```

```

MOV 'array_dtype'      R0, (R5)      : store element from R0
.IFF
.IF      IDN      'array_dtype', D   : array is double
CMPB    dsc$b_dtype(R2), #dsc$k_dtype_dsc : descriptor?
BNEQ    L50
MOVL    4(R2), R0                    : fetch addr of descriptor
MOVB    dsc$b_dtype(R0), dtype(SP)   : load in data type
MOVB    dsc$b_class(R0), class(SP)   : load in class field
MOVAL   data(SP), pointer (SP)
MOVW    #10, str_len(SP)
CMPB    dsc$b_dimct(R2), #1          : check # of dimensions
BNEQ    L52                          : branch if 2 dimensions
PUSHL   R3                          : value of 1st index
PUSHL   R2                          : addr of array desc
PUSHAL  value_desc+8(SP)             : addr of value desc
CALLS   #3,G^BAS$STORE_BFA
BRW     L9
L52:    PUSHL   R4                    : value of 2nd index
        PUSHL   R3                    : value of 1st index
        PUSHL   R2                    : addr of array desc
        PUSHAL  value_desc+12(SP)    : addr of value desc
        CALLS   #4,G^BAS$STORE_BFA
        BRW     L9
L50:    CMPB    dsc$b_class(R2), #dsc$k_class_bfa : virtual array?
        BNEQ    L5                    : no
        JSB     G^BAS$STO_FA_'array_dtype'_R8 : call store routine
        BRW     L9                    : done
L5:     BBS     #5, 10(R2), L6         : br if stored col-wise
        CMPB    dsc$b_dimct(R2), #1    : 1 or 2 dim?
        BNEQ    L51                  : 2 dims
        MOVZWL  dsc$w_length(R2), R6   : make length longword
        INDEX   R3, dsc$l_l1_1(R2), dsc$l_u1_1(R2), R6, #0, R5
        ADDL    dsc$a_a0(R2), R5      : add start addr to offset
        MOV 'array_dtype'      R0, (R5) : store element from R0
        BRW     L9                    : 1 dim done
L51:    INDEX   R3, dsc$l_l1_2(R2), dsc$l_u1_2(R2), dsc$l_m2(R2), #0, R5
        : J * M2
        MOVZWL  dsc$w_length(R2), R6   : longword length for INDEX
        INDEX   R4, dsc$l_l2_2(R2), dsc$l_u2_2(R2), R6, R5, R5
        : (J + (I * M2)) * length
        ADDL    dsc$a_a0(R2), R5      : compute addr of element
        MOV 'array_dtype'      R0, (R5) : store element from R0
        BRW     L9                    : done
L6:     CMPB    dsc$b_dimct(R2), #1    : 1 or 2 dim?
        BNEQ    L61                  : 2 dims
        MOVZWL  dsc$w_length(R2), R6   : make length longword
        INDEX   R4, dsc$l_l1_1(R2), dsc$l_u1_1(R2), R6, #0, R5
        ADDL    dsc$a_a0(R2), R5      : add start addr to offset
        MOV 'array_dtype'      R0, (R5) : store element from R0
        BRW     L9                    : 1 dim done
L61:    INDEX   R4, dsc$l_l2_2(R2), dsc$l_u2_2(R2), dsc$l_m1(R2), #0, R5
        : J * M1
        MOVZWL  dsc$w_length(R2), R6   : longword length for INDEX
        INDEX   R3, dsc$l_l1_2(R2), dsc$l_u1_2(R2), R6, R5, R5
        : (I + (J * M1)) * length
        ADDL    dsc$a_a0(R2), R5      : compute addr of element

```

```

MOV'array_dtype'      R0, (R5)      ; store element from R0
.IFF
CMPB   dsc$b_dtype(R1), #dsc$k_dtype_dsc ; array type other than double
BNEQ   L70
MOVL   4(R1), R0                  ; fetch addr of descriptor
MOVB   dsc$b_dtype(R0), dtype(SP)   ; load in data type
MOVB   dsc$b_class(R0), class(SP)   ; load in class field
MOVAL  data(SP), pointer(SP)
MOVW   #10, str_len(SP)
CMPB   dsc$b_dimct(R1), #1         ; check # of dimensions
BNEQ   L72                        ; branch if 2 dimensions
PUSHL  R2                          ; value of 1st index
PUSHL  R1                          ; addr of array desc
PUSHAL value_desc+8(SP)            ; addr of value desc
CALLS  #3,G^BAS$STORE_BFA
BRW    L9
L72:  PUSHL  R3                      ; value of 2nd index
      PUSHL  R2                      ; value of 1st index
      PUSHL  R1                      ; addr of array desc
      PUSHAL value_desc+12(SP)       ; addr of value desc
      CALLS  #4,G^BAS$STORE_BFA
BRW    L9
L70:  CMPB   dsc$b_class(R1), #dsc$k_class_bfa ;virtual array?
BNEQ   L7
JSB    G^BAS$STO_FA_'array_dtype'_R8 ; call store routine
BRW    L9                            ; done
L7:   BBS    #5, 10(R1), L8           ; br if stored col-wise
      CMPB   dsc$b_dimct(R1), #1     ; 1 or 2 dim?
      BNEQ   L71                    ; 2 dims
      MOVZWL dsc$w_length(R1), R5    ; make length longword
      INDEX  R2, dsc$l_l1_1(R1), dsc$l_u1_1(R1), R5, #0, R4
      ADDL   dsc$a_a0(R1), R4        ; add start addr to offset
      MOV'array_dtype'      R0, (R4) ; store element from R0
      BRW    L9                      ; 1 dim done
L71:  INDEX  R2, dsc$l_l1_2(R1), dsc$l_u1_2(R1), dsc$l_m2(R1), #0, R4
      ; I * M2
      MOVZWL dsc$w_length(R1), R5    ; longword length for INDEX
      INDEX  R3, dsc$l_l2_2(R1), dsc$l_u2_2(R1), R5, R4, R4
      ; (J + (I * M2)) * length
      ADDL   dsc$a_a0(R1), R4        ; compute addr of element
      MOV'array_dtype'      R0, (R4) ; store element from R0
      BRW    L9                      ; done
L8:   CMPB   dsc$b_dimct(R1), #1     ; 1 or 2 dim?
      BNEQ   L81                    ; 2 dims
      MOVZWL dsc$w_length(R1), R5    ; make length longword
      INDEX  R3, dsc$l_l1_1(R1), dsc$l_u1_1(R1), R5, #0, R4
      ADDL   dsc$a_a0(R1), R4        ; add start addr to offset
      MOV'array_dtype'      R0, (R4) ; store element from R0
      BRW    L9                      ; 1 dim done
L81:  INDEX  R3, dsc$l_l2_2(R1), dsc$l_u2_2(R1), dsc$l_m1(R1), #0, R4
      ; J * M1
      MOVZWL dsc$w_length(R1), R5    ; longword length for INDEX
      INDEX  R2, dsc$l_l1_2(R1), dsc$l_u1_2(R1), R5, R4, R4
      ; (I + (J * M1)) * length
      ADDL   dsc$a_a0(R1), R4        ; compute addr of element
      MOV'array_dtype'      R0, (R4) ; store element from R0

```

MATRIX.MAR;1

.ENDC
.ENDC
.ENDC

L9:

.ENDM

