```
BBBBBBBBBBBB        AAAAAAAAA        SSSSSSSSSSS  RRRRRRRRRRR     TTTTTTTTTTTTTTT  LLL
BBBBBBBBBBBB        AAAAAAAAA        SSSSSSSSSSS  RRRRRRRRRRR     TTTTTTTTTTTTTTT  LLL
BBBBBBBBBBBB        AAAAAAAAA        SSSSSSSSSSS  RRRRRRRRRRR     TTTTTTTTTTTTTTT  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
BBBBBBBBBBBB      AAA         AAA     SSSSSSSSS  RRRRRRRRRRR             TTT        LLL
BBBBBBBBBBBB      AAA         AAA     SSSSSSSSS  RRRRRRRRRRR             TTT        LLL
BBBBBBBBBBBB      AAA         AAA     SSSSSSSSS  RRRRRRRRRRR             TTT        LLL
BBB         BBB   AAAAAAAAAAAAAAA           SSS  RRR   RRR               TTT        LLL
BBB         BBB   AAAAAAAAAAAAAAA           SSS  RRR   RRR               TTT        LLL
BBB         BBB   AAAAAAAAAAAAAAA           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
BBBBBBBBBBBB      AAA         AAA   SSSSSSSSSSS     RRR         RRR      TTT        LLLLLLLLLLLLLLL
BBBBBBBBBBBB      AAA         AAA   SSSSSSSSSSS     RRR         RRR      TTT        LLLLLLLLLLLLLLL
BBBBBBBBBBBB      AAA         AAA   SSSSSSSSSSS     RRR         RRR      TTT        LLLLLLLLLLLLLLL
```


BBBBBBBB AAAAAA SSSSSSSS MM MM AAAAAA TTTTTTTTTT SSSSSSSS CCCCCCCC AAAAAA
BBBBBBBB AAAAAA SSSSSSSS MM MM AAAAAA TTTTTTTTTT SSSSSSSS CCCCCCCC AAAAAA
BB BB AA AA SS MMMM MMMM AA AA TT SS CC AA AA
BB BB AA AA SS MMMM MMMM AA AA TT SS CC AA AA
BB BB AA AA SS MM MM MM AA AA TT SS CC AA AA
BB BB AA AA SS MM MM MM AA AA TT SS CC AA AA
BBBBBBBB AA AA SSSSSS MM MM AA AA TT SSSSSS CC AA AA
BBBBBBBB AA AA SSSSSS MM MM AA AA TT SSSSSS CC AA AA
BB BB AAAAAAAAAA SS MM MM AAAAAAAAAA TT SS CC AAAAAAAAAA
BB BB AAAAAAAAAA SS MM MM AAAAAAAAAA TT SS CC AAAAAAAAAA
BB BB AA AA SS MM MM AA AA TT SS CC AA AA ....
BB BB AA AA SS MM MM AA AA TT SS CC AA AA ....
BBBBBBBB AA AA SSSSSSSS MM MM AA AA TT SSSSSSSS CCCCCCCC AA AA ....
BBBBBBBB AA AA SSSSSSSS MM MM AA AA TT SSSSSSSS CCCCCCCC AA AA ....
LL IIIIII SSSSSSSS
LL II SS
LL II SS
LL II SS
LL II SS
LL II SSSSSS
LL II SSSSSS
LL II SS
LL II SS
LL II SS
LL II SS
LLLLLLLLLL IIIIII SSSSSSSS
LLLLLLLLLL IIIIII SSSSSSSS

Table of contents

```
0000     1          .TITLE  BAS$MAT_SCA_MUL          ; Matrix multiply by a scalar
0000     2          .IDENT  /1-025/                  ; File: BASMATSCA.MAR   Edit: DG1025
0000     3
0000     4 ;
0000     5 ;*****************************************************************************
0000     6 ;*                                                                           *
0000     7 ;*  COPYRIGHT (c) 1978, 1980, 1982, 1984 BY                                  *
0000     8 ;*  DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.                   *
0000     9 ;*  ALL RIGHTS RESERVED.                                                     *
0000    10 ;*                                                                           *
0000    11 ;*  THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED    *
0000    12 ;*  ONLY IN  ACCORDANCE WITH  THE  TERMS  OF  SUCH  LICENSE  AND WITH THE   *
0000    13 ;*  INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR  ANY  OTHER    *
0000    14 ;*  COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY    *
0000    15 ;*  OTHER PERSON.  NO TITLE TO AND OWNERSHIP OF  THE  SOFTWARE IS  HEREBY    *
0000    16 ;*  TRANSFERRED.                                                             *
0000    17 ;*                                                                           *
0000    18 ;*  THE INFORMATION IN THIS SOFTWARE IS  SUBJECT TO CHANGE WITHOUT NOTICE    *
0000    19 ;*  AND  SHOULD  NOT  BE  CONSTRUED AS  A COMMITMENT BY DIGITAL EQUIPMENT   *
0000    20 ;*  CORPORATION.                                                             *
0000    21 ;*                                                                           *
0000    22 ;*  DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE  OR  RELIABILITY OF ITS    *
0000    23 ;*  SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.                  *
0000    24 ;*                                                                           *
0000    25 ;*                                                                           *
0000    26 ;*****************************************************************************
0000    27 ;
0000    28
0000    29 ;++
0000    30 ; FACILITY: BASIC code support
0000    31 ;
0000    32 ; ABSTRACT:
0000    33 ;
0000    34 ;       This module multiplies each element of an input array by a scalar.
0000    35 ;
0000    36 ; ENVIRONMENT: User Mode, AST Reentrant
0000    37 ;
0000    38 ; AUTHOR: R. Will, CREATION DATE: 25-Jun-79
0000    39 ;
0000    40 ; MODIFIED BY:
0000    41 ;--
0000    42 ; 1-001 - Original
0000    43 ; 1-002 - Use correct entry point to FLOOR.  RW  24-JUN-79
0000    44 ; 1-003 - Change MTH$DFLOOR_R1 to MTH$DFLOOR_R3.  JBS 25-JUL-1979
0000    45 ; 1-004 - Fix bug with mixed data types.  RW  17-Sept-79
0000    46 ; 1-005 - Redo scaling.  RW  13-Dec-1979
0000    47 ; 1-006 - Change MTH$DFLOOR_R3 to MTH$DINT_R4.  JBS 19-DEC-1979
0000    48 ; 1-007 - Fixed the bug about multiply a double floating by a matrix with scale>0.
0000    49 ;         FM 4-FEB-81.
0000    50 ; 1-008 - Add support for byte, g and h floating.  PLL 18-Sep-81
0000    51 ; 1-009 - More modifications for new data types.  PLL 24-Sep-81
0000    52 ; 1-010 - Change shared external references to G^ RNH 25-Sep-81
0000    53 ; 1-011 - Substitute a macro for the calls to the array fetch and store
0000    54 ;         routines.  This should speed things up.  PLL 9-Nov-81
0000    55 ; 1-012 - STORE macro must handle g & h floating.  PLL 11-Nov-81
0000    56 ; 1-013 - If the scalar is hfloat, the stack will be initialized correctly
0000    57 ;         now.  PLL 17-Nov-81
```

```
0000     58 ; 1-014 - Correct a run-time expression in the FETCH and STORE macros.
0000     59 ;         PLL 20-Jan-82
0000     60 ; 1-015 - Correct bug in BAS$MAT_SC_MUL macro.  (The scalar was not being
0000     61 ;         stored if reals and integers were mixed.)  PLL 15-Feb-82
0000     62 ; 1-016 - Correct FETCH, STORE again.  PLL 23-Feb-82
0000     63 ; 1-017 - Don't list macro expansions.  PLL 16-Mar-82
0000     64 ; 1-018 - Correct STORE_HFLOAT (index off of R4 before changing it). PLL 8-Apr-82
0000     65 ; 1-019 - Remove FETCH and STORE macros; they are now located in macro
0000     66 ;         library MATRIXMAC.OLB.  LB 19-May-82
0000     67 ; 1-020 - Change own storage to stack storage.  LB 9-Jul-1982
0000     68 ; 1-021 - DEST_x_TO_D's which call MTH$DINT_R4 must save and restore R4.
0000     69 ;         (x will be floating data types)  PLL 19-Aug-82
0000     70 ; 1-022 - added in some pound signs where they were missing and causing
0000     71 ;         memory management violations (notably in CMPB's where we were
0000     72 ;         comparing against DSC$K_DTYPE_G and H).  MDL 6-Oct-1982
0000     73 ;         Also allow gfloat results to be stored in a double destination,
0000     74 ;         and vice versa.  PLL 7-Oct-1982
0000     75 ; 1-023 - Use G^ for ALL externals.  SBL 16-Nov-1982
0000     76 ; 1-024 - Redefine stack offsets for the call to the STORE macro (the
0000     77 ;         BSBW adds 4 to the stack).  DG 11-Jan-1984
0000     78 ; 1-025 - Redefine stack offset for 'save_scalar' - the correct offsets
0000     79 ;         from 1-024 are causing scale to be written over by save_scalar
0000     80 ;         in the cases of grand and huge.  DG 9-Apr-1984
0000     81 ;--
```

```
0000     83          .SBTTL  DECLARATIONS
0000     84 ;
0000     85 ; INCLUDE FILES:
0000     86 ;
0000     87
0000     88          $DSCDEF
0000     89          $SFDEF
0000     90
0000     91 ;
0000     92 ; EXTERNAL DECLARATIONS:
0000     93 ;
0000     94
0000     95          .DSABL  GBL                             ; Prevent undeclared
0000     96                                                  ;  symbols from being
0000     97                                                  ;  automatically global.
0000     98          .EXTRN  BAS$K_ARGDONMAT                 ; signalled if all 3 blocks
0000     99                                                  ;  not present in array desc
0000    100                                                  ;  or dimct = 0
0000    101          .EXTRN  BAS$K_DATTYPERR                 ; signalled if dtype of array
0000    102                                                  ; isn't word long float double
0000    103          .EXTRN  BAS$STO_FA_B_R8                 ; array element store for byte
0000    104          .EXTRN  BAS$STO_FA_W_R8                 ; array element store for word
0000    105          .EXTRN  BAS$STO_FA_L_R8                 ; array element store for long
0000    106          .EXTRN  BAS$STO_FA_F_R8                 ; array element store - float
0000    107          .EXTRN  BAS$STO_FA_D_R8                 ; array element store - double
0000    108          .EXTRN  BAS$STO_FA_G_R8                 ; array element store - gfloat
0000    109          .EXTRN  BAS$STO_FA_H_R8                 ; array element store - hfloat
0000    110          .EXTRN  BAS$FET_FA_B_R8                 ; array element fetch - byte
0000    111          .EXTRN  BAS$FET_FA_W_R8                 ; array element fetch - word
0000    112          .EXTRN  BAS$FET_FA_L_R8                 ; array element fetch - long
0000    113          .EXTRN  BAS$FET_FA_F_R8                 ; array element fetch - float
0000    114          .EXTRN  BAS$FET_FA_D_R8                 ; array element fetch - double
0000    115          .EXTRN  BAS$FET_FA_G_R8                 ; array element fetch - gfloat
0000    116          .EXTRN  BAS$FET_FA_H_R8                 ; array element fetch - hfloat
0000    117          .EXTRN  BAS$MAT_REDIM                   ; check if redimensioning of
0000    118                                                  ;  dest array is necessary, if
0000    119                                                  ;  so, do it
0000    120          .EXTRN  BAS$$SCALE_R1                   ; scale of the double precision
0000    121          .EXTRN  BAS$$STOP                       ; signal fatal errors
0000    122          .EXTRN  MTH$DINT_R4                     ; routine to integerize
0000    123          .EXTRN  BAS$FETCH_BFA
0000    124          .EXTRN  BAS$STORE_BFA
0000    125
0000    126 ;
0000    127 ; MACROS:
0000    128 ;
0000    129
0000    130 ;        $BAS$MAT_SC_MUL see below, defines entire scalar multiply algorithm
0000    131 ;        FETCH           fetch an element from an array (found in macro
0000    132 ;                        library MATRIXMAC.OLB)
0000    133 ;        STORE           store an element into an array (found in macro
0000    134 ;                        library MATRIXMAC.OLB)
0000    135
0000    136 ;
0000    137 ; EQUATED SYMBOLS:
0000    138 ;
0000    139
```

```
00000000  0000   140          lower_bnd2 = 0                          ; stack offset for temp
00000004  0000   141          lower_bnd1 = 4                          ; stack offset for temp
00000008  0000   142          upper_bnd1 = 8                          ; stack offset for temp
0000000C  0000   143          save_scalar = 12                        ; stack offset for scalar
0000001C  0000   144          scale = 28                              ; offset from R4 for scale
00000028  0000   145          value_desc = 40                         ; output descriptor
00000028  0000   146          str_len = 40                            ; length field within desc
0000002A  0000   147          dtype = 42                              ; data type field in desc
0000002B  0000   148          class = 43                              ; class field within desc
0000002C  0000   149          pointer = 44                            ; pointer to data in desc
00000030  0000   150          data = 48                               ; data
00000018  0000   151          dsc$l_l1_1 = 24                         ; desc offset if 1 sub
0000001C  0000   152          dsc$l_u1_1 = 28                         ; desc offset if 1 sub
0000001C  0000   153          dsc$l_l1_2 = 28                         ; desc offset if 2 sub
00000020  0000   154          dsc$l_u1_2 = 32                         ; desc offset if 2 sub
00000024  0000   155          dsc$l_l2_2 = 36                         ; desc offset if 2 sub
00000028  0000   156          dsc$l_u2_2 = 40                         ; desc offset if 2 sub
          0000   157
          0000   158 ;
          0000   159 ; OWN STORAGE:
          0000   160 ;
          0000   161
          0000   162
          0000   163 ;
          0000   164 ; PSECT DECLARATIONS:
          0000   165 ;
 00000000 0000   166          .PSECT _BAS$CODE PIC, USR, CON, REL, LCL, SHR, -
          0000   167                          EXE, RD, NOWRT, LONG
          0000   168
```

```
0000    170 ;+
0000    171 ; This macro contains the looping mechanism for accessing all elements of
0000    172 ; an array.  It also contains all the logic for all the combinations of data
0000    173 ; types and scaling.  A macro is used to make it easy to maintain the parallel
0000    174 ; code for all the different combinations of data types.
0000    175 ;-
0000    176         .MACRO  $BAS$MAT_SC_MUL src_dtype, scalar_dtype ; scalar mult algorithm
0000    177
0000    178 ;+
0000    179 ; Decide what mode the multiplication must be done in and store the scalar
0000    180 ; in that data type.  If the data types of the source array and the scalar
0000    181 ; are the same, store the scalar as is.  Else convert the scalar to a common
0000    182 ; type and store.
0000    183 ;-
0000    184
0000    185         .IF     IDN     scalar_dtype, src_dtype ; src array and scalar are
0000    186                                                 ;  same data type
0000    187         MOV'src_dtype'  @scalar(AP), save_scalar(SP) ; save the scalar as is
0000    188         .IFF                                    ; src arrays different dtype
0000    189         .IF     IDN     src_dtype, H            ; source is hfloat
0000    190         CVT'scalar_dtype'H      @scalar(AP), save_scalar(SP) ; cvt scalar to
0000    191                                                 ;  hfloat & save
0000    192         .IFF
0000    193         .IF     IDN     scalar_dtype, H         ; scalar is hfloat
0000    194         MOVH    @scalar(AP), save_scalar(SP)    ; save the scalar
0000    195         .IFF
0000    196         .IF     IDN     src_dtype, G            ; source is gfloat
0000    197         .IF     DIF     scalar_dtype, D         ; don't mix gfloat & dbl
0000    198         CVT'scalar_dtype'G      @scalar(AP), save_scalar(SP) ; cvt scalar to
0000    199                                                 ;  gfloat & save
0000    200         .IFF
0000    201         CVT'scalar_dtype'H      @scalar(AP), save_scalar(SP) ; promote to hfloat
0000    202                                                 ;  if gfloat & dbl
0000    203         .ENDC
0000    204         .IFF
0000    205         .IF     IDN     scalar_dtype, G         ; scalar is gfloat
0000    206         .IF     DIF     src_dtype, D            ; don't mix gfloat & dbl
0000    207         MOVG    @scalar(AP), save_scalar(SP)    ; save the scalar
0000    208         .IFF
0000    209         CVTGH   @scalar(AP), save_scalar(SP)    ; promote to hfloat if
0000    210                                                 ;  gfloat & dbl
0000    211         .ENDC
0000    212         .IFF
0000    213         .IF     IDN     src_dtype, D            ; source is double
0000    214         CVT'scalar_dtype'D      @scalar(AP), save_scalar(SP) ; cvt scalar to
0000    215                                                 ;  double & save
0000    216         .IFF                                    ; 1st array not double
0000    217         .IF     IDN     scalar_dtype, D         ; if scalar double
0000    218         MOVD    @scalar(AP), save_scalar(SP)    ; save the scalar
0000    219         .IFF                                    ; no double operands try float
0000    220         .IF     IDN     src_dtype, F            ; is src array float
0000    221         CVT'scalar_dtype'F      @scalar(AP), save_scalar(SP) ; make scalar float
0000    222         .IFF                                    ; 1st array not float
0000    223         .IF     IDN     scalar_dtype, F         ; is scalar float
0000    224         MOVF    @scalar(AP), save_scalar(SP)    ; yes-store scalar as is
0000    225         .IFF                                    ; no double or float, try long
0000    226         .IF     IDN     src_dtype, L            ; is src array long
```

```
0000    227          CVT'scalar_dtype'L      @scalar(AP), save_scalar(SP) ; make scalar long
0000    228          .IFF
0000    229          .IF     IDN     scalar_dtype, L          ; scalar is long
0000    230          MOVL    @scalar(AP), save_scalar(SP)     ; save scalar as is
0000    231          .IFF
0000    232          .IF     IDN     src_dtype, W             ; source is word
0000    233          CVT'scalar_dtype'W      @scalar(AP), save_scalar(SP) ; make scalar word
0000    234          .IFF
0000    235          .IF     IDN     scalar_dtype, W          ; scalar is word
0000    236          MOVW    @scalar(AP), save_scalar(SP)     ; save scalar as is
0000    237          .IFF
0000    238          .IF     IDN     src_dtype, B             ; source is byte
0000    239          CVT'scalar_dtype'B      @scalar(AP), save_scalar(SP) ; make scalar byte
0000    240          .IFF
0000    241          .IF     IDN     scalar_dtype, B          ; scalar is byte
0000    242          MOVB    @scalar(AP), save_scalar(SP)     ; save scalar as is
0000    243          .IFF                                     ; none of the above - unsupported
0000    244          BRW     ERR_DATTYPERR
0000    245          .ENDC
0000    246          .ENDC
0000    247          .ENDC
0000    248          .ENDC
0000    249          .ENDC
0000    250          .ENDC
0000    251          .ENDC
0000    252          .ENDC
0000    253          .ENDC
0000    254          .ENDC
0000    255          .ENDC
0000    256          .ENDC
0000    257          .ENDC
0000    258          .ENDC
0000    259          .ENDC
0000    260
0000    261 ;+
0000    262 ; Loop through all the rows.  Row and column upper and lower bounds have been
0000    263 ; initialized on the stack.
0000    264 ;-
0000    265
0000    266 LOOP_1ST_SUB'scalar_dtype'src_dtype':
0000    267          MOVL    lower_bnd2(SP), R11              ; R11 has 2nd lower bound
0000    268
0000    269 ;+
0000    270 ; Loop through all the elements (columns) of the current row.  Column lower
0000    271 ; bound is initialized in R11.  Column upper bound is on the stack.
0000    272 ; Distinguish array by data type so that the correct fetch routine can
0000    273 ; retrieve the data, the correct conversion can be done and the correct
0000    274 ; store routine can be called.
0000    275 ;-
0000    276
0000    277 LOOP_2ND_SUB'scalar_dtype'src_dtype':
0000    278
0000    279 ;+
0000    280 ; Get the data from source array
0000    281 ;-
0000    282
0000    283          MOVL    src_matrix(AP), R0               ; pointer to array dest
```

```
0000    284          MOVL    lower_bnd1(SP), R1                 ; current row
0000    285          MOVL    R11, R2                            ; current col
0000    286          FETCH   'src_dtype'                        ; fetch data from src array
0000    287
0000    288 ;+
0000    289 ; Multiply the source by the scalar.
0000    290 ; If the data types of source array and scalar are the same, do the arithmetic
0000    291 ; in that data type.  Else convert the src data to the type that scalar was
0000    292 ; stored in above, and multiply.
0000    293 ; Scaling is needed if the scalar or the src array or both is
0000    294 ; double.
0000    295 ;-
0000    296
0000    297          MOVL    SP, R4                             ; point to stack temps for
0000    298                                                     ;  JSB to storage routine
0000    299          .IF     IDN     scalar_dtype, src_dtype    ; src and scalar are same dtype
0000    300          MUL'src_dtype'2 save_scalar(SP), R0        ; multiply by the scalar
0000    301          .IF     IDN     scalar_dtype, D            ; if both are double
0000    302          DIVD2   scale(SP), R0                      ;  get rid of extra scale
0000    303          CMPD    scale(SP), #1                      ; if scale factor is 0
0000    304          BEQL    1$                                 ;  don't integerize
0000    305          JSB     G^MTH$DINT_R4                      ; else, integerize
0000    306          MOVL    SP, R4                             ; We modified R4 in the above
0000    307                                                     ; JSB, so reload R4.
0000    308          .ENDC
0000    309 1$:      BSBW    DEST_CASE_'src_dtype'              ; store in the destination
0000    310          .IFF                                       ; src & scalar different dtype
0000    311          .IF     IDN     src_dtype, H               ; source is hfloat
0000    312          MULH2   save_scalar(SP), R0                ; scalar was saved hfloat
0000    313          BSBW    DEST_CASE_H                        ; cvt to dest type
0000    314          .IFF
0000    315          .IF     IDN     scalar_dtype, H            ; scalar is hfloat
0000    316          CVT'src_dtype'H R0, R0                     ; make src hfloat
0000    317          MULH2   save_scalar(SP), R0                ; compute product
0000    318          BSBW    DEST_CASE_H                        ; cvt to dest type
0000    319          .IFF
0000    320          .IF     IDN     src_dtype, G               ; source is gfloat
0000    321          .IF     DIF     scalar_dtype, D            ; don't mix gfloat & dbl
0000    322          MULG2   save_scalar(SP), R0                ; mult, scalar is gfloat
0000    323          BSBW    DEST_CASE_G                        ; cvt to dest type
0000    324          .IFF                                       ; scalar dtype is dbl
0000    325                                                     ;  scalar was promoted to hfloat
0000    326          CVTGH   R0, R0                             ; promote src to hfloat
0000    327          MULH2   save_scalar(SP), R0                ; compute product
0000    328          BSBW    DEST_CASE_H                        ; cvt to dest type
0000    329          .ENDC
0000    330          .IFF
0000    331          .IF     IDN     scalar_dtype, G            ; scalar is gfloat
0000    332          .IF     DIF     src_dtype, D               ; don't mix gfloat & dbl
0000    333          CVT'src_dtype'G R0, R0                     ; make src gfloat
0000    334          MULG2   save_scalar(SP), R0                ; compute product
0000    335          BSBW    DEST_CASE_G                        ; cvt to dest type
0000    336          .IFF                                       ; src dtype is dbl
0000    337                                                     ;  scalar was promoted to hfloat
0000    338          CVTDH   R0, R0                             ; cvt src to hfloat
0000    339          MULH2   save_scalar(SP), R0                ; compute product
0000    340          BSBW    DEST_CASE_H                        ; cvt to dest type
```

```
0000          .ENDC
0000          .IFF
0000          .IF     IDN     src_dtype, D            ; source is double =>
0000                                                  ;  scalar has been saved double
0000          MULD2   save_scalar(SP), R0             ; multiply, don't need to descale
0000                                                  ;  because scalar wasn't scaled
0000                                                  ;  don't need to integerize
0000                                                  ;  becausue scalar was W or L
0000          BSBW    DEST_CASE_D                     ; cvrt double product to dest type
0000          .IFF                                    ; 1st array not double
0000          .IF     IDN     scalar_dtype, D         ; is scalar double
0000          CVT'src_dtype'D R0, R0                  ; yes, make src double, is L or W
0000                                                  ;  so don't need integerize, don;t
0000                                                  ;  scale so don't need to descale
0000          MULD2   save_scalar(SP), R0             ; compute the product
0000          BSBW    DEST_CASE_D                     ; cvrt double product to dest type
0000          .IFF                                    ; no double operands try float
0000          .IF     IDN     src_dtype, F            ; src element float =>
0000                                                  ;  scalar has been stored float
0000          MULF2   save_scalar(SP), R0             ; multiply
0000          BSBW    DEST_CASE_F                     ; cvrt float product to dest type
0000          .IFF                                    ; 1st array not float
0000          .IF     IDN     scalar_dtype, F         ; is scalar float
0000          CVT'src_dtype'F R0, R0                  ; yes-make src float
0000          MULF2   save_scalar(SP), R0             ; multiply
0000          BSBW    DEST_CASE_F                     ; cvrt float product to dest type
0000          .IFF                                    ; no double or float, try long
0000          .IF     IDN     src_dtype, L            ; src array long => scalar long
0000          MULL2   save_scalar(SP), R0             ; multiply
0000          BSBW    DEST_CASE_L                     ; convrt long product to dest type
0000          .IFF
0000          .IF     IDN     scalar_dtype, L         ; scalar is long
0000          CVT'src_dtype'L R0, R0                  ; cvt src to long
0000          MULL2   save_scalar(SP), R0             ; multiply
0000          BSBW    DEST_CASE_L                     ; convrt long product to dest type
0000          .IFF
0000          .IF     IDN     src_dtype, W            ; source is word
0000          MULW2   save_scalar(SP), R0             ; mult, scalar is word
0000          BSBW    DEST_CASE_W                     ; cvt to dest type
0000          .IFF
0000          .IF     IDN     scalar_dtype, W         ; scalar is word
0000          CVT'src_dtype'W R0, R0                  ; cvt src to word
0000          MULW2   save_scalar(SP), R0             ; compute product
0000          BSBW    DEST_CASE_W                     ; cvt to dest type
0000          .IFF
0000          .IF     IDN     src_dtype, B            ; scalar is byte
0000          MULB2   save_scalar(SP), R0             ; mult, src is byte
0000          BSBW    DEST_CASE_B                     ; cvt to dest type
0000          .IFF
0000          CVT'src_dtype'B R0, R0                  ; only case left
0000          MULB2   save_scalar(SP), R0             ; compute product
0000          BSBW    DEST_CASE_B
0000          .ENDC
0000          .ENDC
0000          .ENDC
0C00          .ENDC
0000          .ENDC
```

```
0000    398          .ENDC
0000    399          .ENDC
0000    400          .ENDC
0000    401          .ENDC
0000    402          .ENDC
0000    403          .ENDC
0000    404          .ENDC
0000    405          .ENDC
0000    406          .ENDC
0000    407
0000    408 ;+
0000    409 ; Product has now been stored in the destination array.  Continue looping
0000    410 ;-
0000    411
0000    412          INCL    R11                                     ; get next column
0000    413          CMPL    R11, R9                                 ; see if last column done
0000    414          BGTR    3$
0000    415          BRW     LOOP_2ND_SUB'scalar_dtype'src_dtype' ; no, continue inner loop
0000    416
0000    417 ;+
0000    418 ; Have completed entire row.  See if it was the last row.  If not,
0000    419 ; continue with next row.
0000    420 ;-
0000    421
0000    422 3$:      INCL    lower_bnd1(SP)                          ; get next row
0000    423          CMPL    lower_bnd1(SP), upper_bnd1(SP)  ; see if last row done
0000    424          BGTR    5$
0000    425          BRW     LOOP_1ST_SUB'scalar_dtype'src_dtype' ; no, continue outer loop
0000    426
0000    427 5$:      RET                                             ; yes, finished
0000    428
0000    429          .ENDM
```

```
                     0000   431          .SBTTL  BAS$MAT_SCA_MUL  - Multiply each element of an array by scalar
                     0000   432 ;++
                     0000   433 ; FUNCTIONAL DESCRIPTION:
                     0000   434 ;
                     0000   435 ;       This routine multiplies each element of an input matrix by a scalar,
                     0000   436 ;       and stores it in the output matrix.  The algorithm is the same for all
                     0000   437 ;       supported BASIC data types.  In order to keep the code for all
                     0000   438 ;       data type combinations the same and to simplify the reading, the code
                     0000   439 ;       has been done as a macro, which all the data types use varying only
                     0000   440 ;       the letter (B, W, L, F, D, G, H) in multiplying by the scalar, converting fr
                     0000   441 ;       source to dest type, and calling the array store routines.
                     0000   442 ;       The scalar must be the same datatype as the source matrix.
                     0000   443 ;
                     0000   444 ; CALLING SEQUENCE:
                     0000   445 ;
                     0000   446 ;       CALL BAS$MAT_SCA_MUL (scalar.rz.r, scalar_dtype.rlu.v,
                     0000   447 ;                             srcmatrix.rx.da, destmatrix.wx.da)
                     0000   448 ;
                     0000   449 ; INPUT PARAMETERS:
                     0000   450 ;
00000004             0000   451          scalar = 4      ; pointer to scalar of type specified in next parameter
00000008             0000   452          scalar_dtype = 8 ; VAX standard dtype code for data type of scalar
0000000C             0000   453          src_matrix = 12 ; pointer to descriptor for source matrix
                     0000   454 ;
                     0000   455 ; IMPLICIT INPUTS:
                     0000   456 ;
                     0000   457 ;       scale from callers BASIC frame if data type of either matrix is double
                     0000   458 ;
                     0000   459 ; OUTPUT PARAMETERS:
                     0000   460 ;
00000010             0000   461          dest_matrix = 16        ; pointer to descriptor for destination matrix
                     0000   462 ;
                     0000   463 ; IMPLICIT OUTPUTS:
                     0000   464 ;
                     0000   465 ;       NONE
                     0000   466 ;
                     0000   467 ; FUNCTION VALUE:
                     0000   468 ; COMPLETION CODES:
                     0000   469 ;
                     0000   470 ;       NONE
                     0000   471 ;
                     0000   472 ; SIDE EFFECTS:
                     0000   473 ;
                     0000   474 ;       This routine calls the BASIC matrix fetch, store, and redimensioning
                     0000   475 ;       routines, and may cause any of their errors to be signalled.  It also
                     0000   476 ;       may signal any of the errors listed in the externals area.
                     0000   477 ;       It may also cause the destination array to have different dimensions.
                     0000   478 ;
                     0000   479 ;--
                     0000   480
                 4FFC 0000   481          .ENTRY BAS$MAT_SCA_MUL, ^M<R2,R3,R4,R5,R6,R7,R8,R9,R10,R11,IV>
                     0002   482
                     0002   483 ;+
                     0002   484 ;       REGISTER USAGE
                     0002   485 ;       R0 - R8 destroyed by store routines
                     0002   486 ;       R9      upper bound for 2nd subscript
                     0002   487 ;       R10     pointer to destination array descriptor
```

```
                                0002   488 ;       R11     current value of 2nd subscript
                                0002   489 ;-
                                0002   490
                                0002   491 ;+
                                0002   492 ; Put routine arguments into registers for ease of use.
                                0002   493 ; If block 2 of array descriptor (multipliers) is not present then error.
                                0002   494 ;-
                                0002   495
      52    0C AC    D0        0002   496         MOVL    src_matrix(AP), R2              ; ptr to array descr in R2
  23 0A A2    07    E1        0006   497         BBC     #DSC$V_FL_BOUNDS, DSC$B_AFLAGS(R2), ERR_ARGDONMAT
                                000B   498                                                 ; exit if block 3 not
                                000B   499                                                 ; present in descriptor
      5A    10 AC    D0        000B   500         MOVL    dest_matrix(AP), R10            ; ptr to dest array descriptor
      50    0C AD    D0        000F   501         MOVL    SF$L_SAVE_FP(FP), R0            ; pass FP to get scale
00000000'GF    16    0013   502         JSB     G^BAS$$SCALE_R1                 ; get scale in R0 & R1
                                0019   503                                                 ; call a BLISS routine because
                                0019   504                                                 ; the frame offsets are only
                                0019   505                                                 ; defined for BLISS
            7E    7C    0019   506         CLRQ    -(SP)                           ; save space for VALUE_DESC
            7E    7C    001B   507         CLRQ    -(SP)                           ;       AND
            7E    7C    001D   508         CLRQ    -(SP)                           ; DATA
         7E    50    70    001F   509         MOVD    R0, -(SP)                       ; save the scale
            7E    7C    0022   510         CLRQ    -(SP)                           ; save place for scalar on stack
            7E    7C    0024   511         CLRQ    -(SP)                           ;  scalar may be hfloat
                                0026   512 ;+
                                0026   513 ; Set up limits for looping through all elements
                                0026   514 ;-
                                0026   515
   01    0B A2    91    0026   516         CMPB    DSC$B_DIMCT(R2), #1             ; determine # of subscripts
         0F    13    002A   517         BEQLU   INIT_ONE_SUB                    ; 1 sub, go init
         2B    1A    002C   518         BGTRU   INIT_TWO_SUBS                   ; >=2 subs, go init
                                002E   519                                                 ; 0 subs, fall into error proc
                                002E   520
                                002E   521 ERR_ARGDONMAT:
   00000000'8F    DD    002E   522         PUSHL   #BAS$K_ARGDONMAT                ; signal error, 0 for dimct
00000000'GF    01    FB    0034   523         CALLS   #1, G^BAS$$STOP                 ; or block 2 or 3 absent
                                003B   524
                                003B   525 ;+
                                003B   526 ; There is only 1 subscript.  Redimension the destination array.
                                003B   527 ; Make both upper and lower bound for 2nd
                                003B   528 ; subscript a 1.  A second subscript will be passed to and ignored by the
                                003B   529 ; store routine.  Put bounds for 1st subscript on stack.
                                003B   530 ;-
                                003B   531
                                003B   532 INIT_ONE_SUB:
      1C A2    DD    003B   533         PUSHL   dsc$l_u1_1(R2)                  ; get bound for redim
         5A    DD    003E   534         PUSHL   R10                             ; pointer to dest array desc
00000000'GF    02    FB    0040   535         CALLS   #2, G^BAS$MAT_REDIM             ; redimension the dest
      1C A2    DD    0047   536         PUSHL   dsc$l_u1_1(R2)                  ; 1st upper bound
      18 A2    DD    004A   537         PUSHL   dsc$l_l1_1(R2)                  ; 1st lower bound
         03    14    004D   538         BGTR    1$                              ; not 0 or neg, do 2nd sub
      6E    01    D0    004F   539         MOVL    #1, (SP)                        ; don't alter col 0
         01    DD    0052   540 1$:      PUSHL   #1                              ; dummy 2nd upper bound
      59    01    D0    0054   541         MOVL    #1, R9                          ; dummy 2nd lower bound
         26    11    0057   542         BRB     SEPARATE_DTYPES                 ; go loop
                                0059   543
                                0059   544 ;+
```

```
                                0059   545 ; There are 2 subscripts.  Check and redimension the destination array if
                                0059   546 ; necessary.  Put the upper bound for both subscripts on the
                                0059   547 ; stack and make sure that the lower bound for both subscripts will start
                                0059   548 ; at 1 (do not alter row or col 0)
                                0059   549 ;-
                                0059   550
                                0059   551 INIT_TWO_SUBS:
                          28 A2   DD    0059   552         PUSHL   dsc$l_u2_2(R2)                  ; 2nd upper bound
                          20 A2   DD    005C   553         PUSHL   dsc$l_u1_2(R2)                  ; 1st upper bound
                             5A   DD    005F   554         PUSHL   R10                             ; dest array pointer
              00000000'GF    03   FB    0061   555         CALLS   #3, G^BAS$MAT_REDIM             ; redimension destination
                          20 A2   DD    0068   556         PUSHL   dsc$l_u1_2(R2)                  ; 1st upper bound
                          1C A2   DD    006B   557         PUSHL   dsc$l_l1_2(R2)                  ; 1st lower bound
                             03   14    006E   558         BGTR    1$                              ; not row 0 or neg, do cols
                          6E   01   D0    0070   559         MOVL    #1, (SP)                        ; start with row 1
                       59 28 A2   D0    0073   560 1$:      MOVL    dsc$l_u2_2(R2), R9              ; 2nd upper bound
                          24 A2   DD    0077   561         PUSHL   dsc$l_l2_2(R2)                  ; 2nd lower bound
                             03   14    007A   562         BGTR    SEPARATE_DTYPES                 ; not col 0 or neg, go loop
                          6E   01   D0    007C   563         MOVL    #1, (SP)                        ; start with col 1
                                0007F   564
                                0007F   565 ;+
                                0007F   566 ; Algorithm now differs according to source array data types
                                0007F   567 ;-
                                0007F   568
                                0007F   569 SEPARATE_DTYPES:
                    05  06  02 A2   8F    007F   570         CASEB   DSC$B_DTYPE(R2), #DSC$K_DTYPE_B, #<DSC$K_DTYPE_D - DSC$K_DTYPE_B>
                              0037'  0084   571 2$:      .WORD   BYTE-2$                         ; code for byte dtype
                              07E6'  0086   572         .WORD   WORD-2$                         ; code for word dtype
                              0F92'  0088   573         .WORD   LONG-2$                         ; code for long dtype
                              002A'  008A   574         .WORD   ERR_DATTYPERR-2$                ; quad not supported
                              173B'  008C   575         .WORD   FLOAT-2$                        ; code for float dtype
                              1EE1'  008E   576         .WORD   DOUBLE-2$                       ; code for double dtype
                                0090   577
                                0090   578 ;+
                                0090   579 ; G and H floating fall outside the range of the CASEB statement.
                                0090   580 ;-
                                0090   581
                       1B   0C A2   91    0090   582         CMPB    DSC$B_DTYPE(R2), #DSC$K_DTYPE_G
                             03   12    0094   583         BNEQ    3$
                           2682   31    0096   584         BRW     GFLOAT
                                0099   585
                       1C   02 A2   91    0099   586 3$:      CMPB    DSC$B_DTYPE(R2), #DSC$K_DTYPE_H
                             03   12    009D   587         BNEQ    4$
                           2E50   31    009F   588         BRW     HFLOAT
                                00A2   589
                       18   02 A2   91    00A2   590 4$:      CMPB    DSC$B_DTYPE(R2), #DSC$K_DTYPE_DSC
                             06   12    00A6   591         BNEQ    ERR_DATTYPERR
                       52   04 A2   D0    00A8   592         MOVL    4(R2), R2                       ; R2 <-- addr of descriptor
                             D1   11    00AC   593         BRB     SEPARATE_DTYPES                 ; CASE again for dtype in desc
                                00AE   594
                                00AE   595 ERR_DATTYPERR:
                  00000000'8F    DD    00AE   596         PUSHL   #BAS$K_DATTYPERR                ; Signal error, unsupported
              00000000'GF    01   FB    00B4   597         CALLS   #1, G^BAS$$STOP                 ; dtype in array desc
```

```
                                00BB    600 ;+
                                00BB    601 ; Source array is a byte array.  Now differentiate on the scalar type.
                                00BB    602 ;-
                                00BB    603
                     55   5C    D0  00BB    604 BYTE:   MOVL    AP, R5
                05   06   08 A5   8F  00BE    605 5$:     CASEB   scalar_dtype(R5), #DSC$K_DTYPE_B, #<DSC$K_DTYPE_D - DSC$K_DTYPE_B>
                                002D' 00C3    606 1$:     .WORD   BYTE_TO_BYTE-1$                 ; code for byte dtype
                                013B' 00C5    607         .WORD   BYTE_TO_WORD-1$                 ; code for word dtype
                                024C' 00C7    608         .WORD   BYTE_TO_LONG-1$                 ; code for long dtype
                                FFEB  00C9    609         .WORD   ERR_DATTYPERR-1$                ; quad not supported
                                035D' 00CB    610         .WORD   BYTE_TO_FLOAT-1$                ; code for float dtype
                                046E' 00CD    611         .WORD   BYTE_TO_DOUBLE-1$               ; code for double dtype
                                      00CF    612
                                      00CF    613 ;+
                                      00CF    614 ; Check for g and h floating separately, since they fall outside the range
                                      00CF    615 ; of the CASEB.
                                      00CF    616 ;-
                                      00CF    617
                1B   08 A5   91     00CF    618         CMPB    scalar_dtype(R5), #DSC$K_DTYPE_G
                       03   12     00D3    619         BNEQ    2$
                     056A   31     00D5    620         BRW     BYTE_TO_GFLOAT
                                      00D8    621
                1C   08 A5   91     00D8    622 2$:     CMPB    scalar_dtype(R5), #DSC$K_DTYPE_H
                       03   12     00DC    623         BNEQ    3$
                     0675   31     00DE    624         BRW     BYTE_TO_HFLOAT
                                      00E1    625
                18   08 A5   91     00E1    626 3$:     CMPB    scalar_dtype(R5), #DSC$K_DTYPE_DSC
                       06   12     00E5    627         BNEQ    4$
                55   0C A5   D0     00E7    628         MOVL    SRC_MATRIX(R5), R5
                       D1   11     00EB    629         BRB     5$
                                      00ED    630
                     FFBE   31     00ED    631 4$:     BRW     ERR_DATTYPERR                   ; unsupported dtype
                                      00F0    632 ;+
                                      00F0    633 ; Now type of source array and scalar are known.  Use the macro to
                                      00F0    634 ; generate the code for each case
                                      00F0    635 ;-
                                      00F0    636
```

```
00F0   638 BYTE_TO_BYTE:    $BAS$MAT_SC_MUL B, B
01FE   639
```

```
                        641 BYTE_TO_WORD:   $BAS$MAT_SC_MUL B, W
                        642
```

```
                030F    644 BYTE_TO_LONG:   $BAS$MAT_SC_MUL B, L
                0420    645
```

```
0420    647 BYTE_TO_FLOAT:  $BAS$MAT_SC_MUL B, F
0531    648
```

```
                        0531    650 BYTE_TO_DOUBLE: $BAS$MAT_SC_MUL B, D
                        0642    651
```

```
                        653 BYTE_TO_GFLOAT: $BAS$MAT_SC_MUL B, G
                        654
```

```
                0756    656 BYTE_TO_HFLOAT: $BAS$MAT_SC_MUL B, H
```

```
                                086A   658 ;+
                                086A   659 ; Source array is a word array.  Now differentiate on the scalar type.
                                086A   660 ;-
                                086A   661
                     55    5C   D0   086A   662 WORD:   MOVL    AP, R5
                05   06    08 A5   8F   086D   663 5$:     CASEB   scalar_dtype(R5), #DSC$K_DTYPE_B, #<DSC$K_DTYPE_D - DSC$K_DTYPE_B>
                                002D' 0872   664 1$:     .WORD   WORD_TO_BYTE-1$                 ; code for byte dtype
                                013B' 0874   665         .WORD   WORD_TO_WORD-1$                 ; code for word dtype
                                0249' 0876   666         .WORD   WORD_TO_LONG-1$                 ; code for long dtype
                                F83C  0878   667         .WORD   ERR_DATTYPERR-1$                ; quad not supported
                                035A' 087A   668         .WORD   WORD_TO_FLOAT-1$                ; code for float dtype
                                046B' 087C   669         .WORD   WORD_TO_DOUBLE-1$               ; code for double dtype
                                      087E   670
                                      087E   671 ;+
                                      087E   672 ; Check for g and h floating separately, since they fall outside the range
                                      087E   673 ; of the CASEB statement.
                                      087E   674 ;-
                                      087E   675
                          1B    08 A5   91   087E   676         CMPB    scalar_dtype(R5), #DSC$K_DTYPE_G
                                   03   12   0882   677         BNEQ    2$
                                 0567   31   0884   678         BRW     WORD_TO_GFLOAT
                                      0887   679
                          1C    08 A5   91   0887   680 2$:     CMPB    scalar_dtype(R5), #DSC$K_DTYPE_H
                                   03   12   088B   681         BNEQ    3$
                                 0672   31   088D   682         BRW     WORD_TO_HFLOAT
                                      0890   683
                          18    08 A5   91   0890   684 3$:     CMPB    scalar_dtype(R5), #DSC$K_DTYPE_DSC
                                   06   12   0894   685         BNEQ    4$
                          55    0C A5   D0   0896   686         MOVL    SRC_MATRIX(R5), R5
                                   D1   11   089A   687         BRB     5$
                                      089C   688
                                 F80F   31   089C   689 4$:     BRW     ERR_DATTYPERR                   ; unsupported dtype
                                      089F   690
                                      089F   691 ;+
                                      089F   692 ; Now type of source array and scalar are known.  Use the macro to
                                      089F   693 ; generate the code for each case
                                      089F   694 ;-
                                      089F   695
```

```
089F    697 WORD_TO_BYTE:   $BAS$MAT_SC_MUL W, B
09AD    698
```

```
                09AD    700 WORD_TO_WORD:   $BAS$MAT_SC_MUL W, W
                0ABB    701
```

```
                        703 WORD_TO_LONG:   $BAS$MAT_SC_MUL W, L
                        704
```

```
                        706 WORD_TO_FLOAT:  $BAS$MAT_SC_MUL W, F
OBCC
OCDD                    707
```

```
                        0CDD    709 WORD_TO_DOUBLE: $BAS$MAT_SC_MUL W, D
                        0DEE    710
```

```
ODEE   712 WORD_TO_GFLOAT: $BAS$MAT_SC_MUL W, G
0F02   713
```

```
                0F02    715 WORD_TO_HFLOAT: $BAS$MAT_SC_MUL W, H
                1016    716
```

```
                                1016   718 ;+
                                1016   719 ; Source array is a longword array.  Now differentiate on the scalar type
                                1016   720 ;-
                                1016   721
                     55   5C    D0   1016   722 LONG:   MOVL    AP, R5
                05   06   08 A5   8F   1019   723 5$:     CASEB   scalar_dtype(R5), #DSC$K_DTYPE_B, #<DSC$K_DTYPE_D - DSC$K_DTYPE_B>
                               002D' 101E   724 1$:     .WORD   LONG_TO_BYTE-1$                 ; code for byte dtype
                               013B' 1020   725         .WORD   LONG_TO_WORD-1$                 ; code for word dtype
                               0249' 1022   726         .WORD   LONG_TO_LONG-1$                 ; code for long dtype
                               F090  1024   727         .WORD   ERR_DATTYPERR-1$                ; quad not supported
                               0357' 1026   728         .WORD   LONG_TO_FLOAT-1$                ; code for float dtype
                               0468' 1028   729         .WORD   LONG_TO_DOUBLE-1$               ; code for double dtype
                                     102A   730
                                     102A   731 ;+
                                     102A   732 ; Check for g and h floating separately, since they fall outside the range
                                     102A   733 ; of the CASEB statement.
                                     102A   734 ;-
                                     102A   735
                     1B   08 A5   91   102A   736         CMPB    scalar_dtype(R5), #DSC$K_DTYPE_G
                            03   12   102E   737         BNEQ    2$
                          0564   31   1030   738         BRW     LONG_TO_GFLOAT
                                     1033   739
                     1C   08 A5   91   1033   740 2$:     CMPB    scalar_dtype(R5), #DSC$K_DTYPE_H
                            03   12   1037   741         BNEQ    3$
                          066F   31   1039   742         BRW     LONG_TO_HFLOAT
                                     103C   743
                     18   08 A5   91   103C   744 3$:     CMPB    scalar_dtype(R5), #DSC$K_DTYPE_DSC
                            06   12   1040   745         BNEQ    4$
                     55   0C A5   D0   1042   746         MOVL    SRC_MATRIX(R5), R5
                            D1   11   1046   747         BRB     5$
                                     1048   748
                          F063   31   1048   749 4$:     BRW     ERR_DATTYPERR                   ; unsupported dtype
                                     104B   750
                                     104B   751
                                     104B   752 ;+
                                     104B   753 ; Now type of source array and scalar are known.  Use the macro to
                                     104B   754 ; generate the code for each case
                                     104B   755 ;-
                                     104B   756
```

```
                        758 LONG_TO_BYTE:   $BAS$MAT_SC_MUL L, B
                        759
```

```
1159    761 LONG_TO_WORD:   $BAS$MAT_SC_MUL L, W
1267    762
```

```
     1267    764 LONG_TO_LONG:    $BAS$MAT_SC_MUL L, L
     1375    765
```

```
1375    767 LONG_TO_FLOAT:  $BAS$MAT_SC_MUL L, F
1486    768
```

```
1486    770 LONG_TO_DOUBLE: $BAS$MAT_SC_MUL L, D
1597    771
```

```
                        773 LONG_TO_GFLOAT: $BAS$MAT_SC_MUL L, G
                        774
```

```
                        776 LONG_TO_HFLOAT: $BAS$MAT_SC_MUL L, H
                        777
```

```
                                17BF    779 ;+
                                17BF    780 ; Source array is a floating array.  Now differentiate on the scalar type
                                17BF    781 ;-
                                17BF    782
                     55   5C    D0  17BF    783 FLOAT:  MOVL    AP, R5
                05   06   08 A5    8F  17C2    784 5$:     CASEB   scalar_dtype(R5), #DSC$K_DTYPE_B, #<DSC$K_DTYPE_D - DSC$K_DTYPE_B>
                                002D' 17C7    785 1$:     .WORD   FLOAT_TO_BYTE-1$                ; code for byte dtype
                                013B' 17C9    786         .WORD   FLOAT_TO_WORD-1$                ; code for word dtype
                                0249' 17CB    787         .WORD   FLOAT_TO_LONG-1$                ; code for long dtype
                                E8E7  17CD    788         .WORD   ERR_DATTYPERR-1$                ; quad not supported
                                0357' 17CF    789         .WORD   FLOAT_TO_FLOAT-1$               ; code for float dtype
                                0465' 17D1    790         .WORD   FLOAT_TO_DOUBL-1$               ; code for double dtype
                                      17D3    791
                                      17D3    792 ;+
                                      17D3    793 ; Check for g and h floating separately, since they fall outside the range
                                      17D3    794 ; of the CASEB statement.
                                      17D3    795 ;-
                                      17D3    796
                           1B   08 A5   91  17D3    797         CMPB    scalar_dtype(R5), #DSC$K_DTYPE_G
                                  03   12  17D7    798         BNEQ    2$
                                0561   31  17D9    799         BRW     FLOAT_TO_GFLOA
                                      17DC    800
                           1C   08 A5   91  17DC    801 2$:     CMPB    scalar_dtype(R5), #DSC$K_DTYPE_H
                                  03   12  17E0    802         BNEQ    3$
                                066C   31  17E2    803         BRW     FLOAT_TO_HFLOA
                                      17E5    804
                           18   08 A5   91  17E5    805 3$:     CMPB    scalar_dtype(R5), #DSC$K_DTYPE_DSC
                                  06   12  17E9    806         BNEQ    4$
                           55   0C A5   D0  17EB    807         MOVL    SRC_MATRIX(R5), R5
                                  D1   11  17EF    808         BRB     5$
                                      17F1    809
                                E8BA   31  17F1    810 4$:     BRW     ERR_DATTYPERR                   ; unsupported dtype
                                      17F4    811
                                      17F4    812 ;+
                                      17F4    813 ; Now type of source array and scalar are known.  Use the macro to
                                      17F4    814 ; generate the code for each case
                                      17F4    815 ;-
                                      17F4    816
```

```
                        818 FLOAT_TO_BYTE:  $BAS$MAT_SC_MUL F, B
                        819
```

```
                                I  4

     1902    821 FLOAT_TO_WORD:  $BAS$MAT_SC_MUL F, W
     1A10    822
```

```
                        824 FLOAT_TO_LONG:  $BAS$MAT_SC_MUL F, L
                        825
```

```
                                                K  4

                1B1E    827 FLOAT_TO_FLOAT: $BASSMAT_SC_MUL F, F
                1C2C    828
```

```
                        830 FLOAT_TO_DOUBL: $BAS$MAT_SC_MUL F, D
                        831
```

```
                        833 FLOAT_TO_GFLOA: $BAS$MAT_SC_MUL F, G
                        834
```

```
                1E51    836 FLOAT_TO_HFLOA: $BAS$MAT_SC_MUL F, H
```

```
                                1F65    838 ;+
                                1F65    839 ; Source array is a double array.  Now differentiate on the scalartype.
                                1F65    840 ;-
                                1F65    841
                     55   5C    D0  1F65    842 DOUBLE: MOVL    AP, R5
                05   06   08 A5   8F  1F68    843 5$:     CASEB   scalar_dtype(R5), #DSC$K_DTYPE_B, #<DSC$K_DTYPE_D - DSC$K_DTYPE_B>
                               002D' 1F6D    844 1$:     .WORD   DOUBLE_TO_BYTE-1$               ; code for byte dtype
                               013B' 1F6F    845         .WORD   DOUBLE_TO_WORD-1$               ; code for word dtype
                               0249' 1F71    846         .WORD   DOUBLE_TO_LONG-1$               ; code for long dtype
                               E141  1F73    847         .WORD   ERR_DATTYPERR-1$                ; quad not supported
                               0357' 1F75    848         .WORD   DOUBLE_TO_FLOA-1$               ; code for float dtype
                               0465' 1F77    849         .WORD   DOUBLE_TO_DOUBL-1$              ; code for double dtype
                                1F79    850
                                1F79    851 ;+
                                1F79    852 ; Check for g and h floating separately, since they fall outside the range
                                1F79    853 ; of the CASEB statement.
                                1F79    854 ;-
                                1F79    855
                1B   08 A5   91  1F79    856         CMPB    scalar_dtype(R5), #DSC$K_DTYPE_G
                        03   12  1F7D    857         BNEQ    2$
                      0571   31  1F7F    858         BRW     DOUBLE_TO_GFLOA
                                1F82    859
                1C   08 A5   91  1F82    860 2$:     CMPB    scalar_dtype(R5), #DSC$K_DTYPE_H
                        03   12  1F86    861         BNEQ    3$
                      067C   31  1F88    862         BRW     DOUBLE_TO_HFLOA
                                1F8B    863
                18   08 A5   91  1F8B    864 3$:     CMPB    scalar_dtype(R5), #DSC$K_DTYPE_DSC
                        06   12  1F8F    865         BNEQ    4$
                55   0C A5   D0  1F91    866         MOVL    SRC_MATRIX(R5), R5
                        D1   11  1F95    867         BRB     5$
                                1F97    868
                     E114   31  1F97    869 4$:     BRW     ERR_DATTYPERR                   ; unsupported dtype
                                1F9A    870
                                1F9A    871 ;+
                                1F9A    872 ; Now type of source array and scalar are known.  Use the macro to
                                1F9A    873 ; generate the code for each case
                                1F9A    874 ;-
                                1F9A    875
```

```
1F9A    877 DOUBLE_TO_BYTE: $BAS$MAT_SC_MUL D, B
20A8    878
```

```
                    880 DOUBLE_TO_WORD: $BAS$MAT_SC_MUL D, W
                    881
```

```
                21B6    883 DOUBLE_TO_LONG: $BAS$MAT_SC_MUL D, L
                22C4    884
```

```
                        886 DOUBLE_TO_FLOA: $BAS$MAT_SC_MUL D, F
                        887
```

```
                        889 DOUBLE_TO_DOUBL: $BAS$MAT_SC_MUL D, D
                        890
```

```
                        892 DOUBLE_TO_GFLOA: $BAS$MAT_SC_MUL D, G
                        893
```

```
                        2607    895 DOUBLE_TO_HFLOA: $BAS$MAT_SC_MUL D, H
                        271B    896
```

```
                                271B    898 ;+
                                271B    899 ; Source array is a gfloat array.  Now differentiate on the scalartype.
                                271B    900 ;-
                                271B    901
                     55   5C    D0  271B    902 GFLOAT: MOVL    AP, R5
                05   06   08 A5   8F  271E    903 5$:     CASEB   scalar_dtype(R5), #DSC$K_DTYPE_B, #<DSC$K_DTYPE_D - DSC$K_DTYPE_B>
                                002D' 2723    904 1$:     .WORD   GFLOAT_TO_BYTE-1$               ; code for byte dtype
                                0143' 2725    905         .WORD   GFLOAT_TO_WORD-1$               ; code for word dtype
                                0259' 2727    906         .WORD   GFLOAT_TO_LONG-1$               ; code for long dtype
                                D98B  2729    907         .WORD   ERR_DATTYPERR-1$                ; quad not supported
                                036F' 272B    908         .WORD   GFLOAT_TO_FLOA-1$               ; code for float dtype
                                0485' 272D    909         .WORD   GFLOAT_TO_DOUBL-1$              ; code for double dtype
                                      272F    910
                                      272F    911 ;+
                                      272F    912 ; Check for g and h floating separately, since they fall outside the range
                                      272F    913 ; of the CASEB statement.
                                      272F    914 ;-
                                      272F    915
                1B   08 A5   91     272F    916         CMPB    scalar_dtype(R5), #DSC$K_DTYPE_G
                        03   12     2733    917         BNEQ    2$
                      058A   31     2735    918         BRW     GFLOAT_TO_GFLOA
                                    2738    919
                1C   08 A5   91     2738    920 2$:     CMPB    scalar_dtype(R5), #DSC$K_DTYPE_H
                        03   12     273C    921         BNEQ    3$
                      0697   31     273E    922         BRW     GFLOAT_TO_HFLOA
                                    2741    923
                18   08 A5   91     2741    924 3$:     CMPB    scalar_dtype(R5), #DSC$K_DTYPE_DSC
                        06   12     2745    925         BNEQ    4$
                55   0C A5   D0     2747    926         MOVL    SRC_MATRIX(R5), R5
                        D1   11     274B    927         BRB     5$
                                    274D    928
                      D95E   31     274D    929 4$:     BRW     ERR_DATTYPERR                   ; unsupported dtype
                                    2750    930
                                    2750    931 ;+
                                    2750    932 ; Now type of source array and scalar are known.  Use the macro to
                                    2750    933 ; generate the code for each case
                                    2750    934 ;-
                                    2750    935
```

```
                2750    937 GFLOAT_TO_BYTE: $BAS$MAT_SC_MUL G, B
                2866    938
```

```
                2866    940 GFLOAT_TO_WORD: $BAS$MAT_SC_MUL G, W
                297C    941
```

```
                        943 GFLOAT_TO_LONG: $BAS$MAT_SC_MUL G, L
                        944
```

```
                2A92    946 GFLOAT_TO_FLOA: $BAS$MAT_SC_MUL G, F
                2BA8    947
```

```
                                                B 6

                        2BA8    949 GFLOAT_TO_DOUBL: $BAS$MAT_SC_MUL G, D
                        2CC2    950
```

```
                        952 GFLOAT_TO_GFLOA: $BAS$MAT_SC_MUL G, G
                        953
```

```
                     955 GFLOAT_TO_HFLOA: $BAS$MAT_SC_MUL G, H
                     956
```

```
                                2EF2    958 ;+
                                2EF2    959 ; Source array is an hfloat array.  Now differentiate on the scalartype.
                                2EF2    960 ;-
                                2EF2    961
                     55   5C    D0   2EF2    962 HFLOAT: MOVL    AP, R5
                05   06   08 A5  8F   2EF5    963 5$:     CASEB   scalar_dtype(R5), #DSC$K_DTYPE_B, #<DSC$K_DTYPE_D - DSC$K_DTYPE_B>
                               002D' 2EFA    964 1$:     .WORD   HFLOAT_TO_BYTE-1$               ; code for byte dtype
                               0143' 2EFC    965         .WORD   HFLOAT_TO_WORD-1$               ; code for word dtype
                               0259' 2EFE    966         .WORD   HFLOAT_TO_LONG-1$               ; code for long dtype
                               D1B4  2F00    967         .WORD   ERR_DATTYPERR-1$                ; quad not supported
                               036F' 2F02    968         .WORD   HFLOAT_TO_FLOA-1$               ; code for float dtype
                               0485' 2F04    969         .WORD   HFLOAT_TO_DOUBL-1$              ; code for double dtype
                                2F06    970
                                2F06    971 ;+
                                2F06    972 ; Check for g and h floating separately, since they fall outside the range
                                2F06    973 ; of the CASEB statement.
                                2F06    974 ;-
                                2F06    975
                1B   08 A5   91 2F06    976         CMPB    scalar_dtype(R5), #DSC$K_DTYPE_G
                       03    12 2F0A    977         BNEQ    2$
                     0586    31 2F0C    978         BRW     HFLOAT_TO_GFLOA
                                2F0F    979
                1C   08 A5   91 2F0F    980 2$:     CMPB    scalar_dtype(R5), #DSC$K_DTYPE_H
                       03    12 2F13    981         BNEQ    3$
                     0693    31 2F15    982         BRW     HFLOAT_TO_HFLOA
                                2F18    983
                18   08 A5   91 2F18    984 3$:     CMPB    scalar_dtype(R5), #DSC$K_DTYPE_DSC
                       06    12 2F1C    985         BNEQ    4$
                55   0C A5   D0 2F1E    986         MOVL    SRC_MATRIX(R5), R5
                       D1    11 2F22    987         BRB     5$
                                2F24    988
                   D187      31 2F24    989 4$:     BRW     ERR_DATTYPERR                   ; unsupported dtype
                                2F27    990
                                2F27    991 ;+
                                2F27    992 ; Now type of source array and scalar are known.  Use the macro to
                                2F27    993 ; generate the code for each case
                                2F27    994 ;-
                                2F27    995
```

```
                2F27    997 HFLOAT_TO_BYTE: $BAS$MAT_SC_MUL H, B
                303D    998
```

```
                303D  1000 HFLOAT_TO_WORD: $BAS$MAT_SC_MUL H, W
                3153  1001
```

```
                3153  1003 HFLOAT_TO_LONG: $BAS$MAT_SC_MUL H, L
                3269  1004
```

```
                3269  1006 HFLOAT_TO_FLOA: $BAS$MAT_SC_MUL H, F
                337F  1007
```

```
                        1009 HFLOAT_TO_DOUBL: $BAS$MAT_SC_MUL H, D
                        1010
```

```
                        3495  1012 HFLOAT_TO_GFLOA: $BAS$MAT_SC_MUL H, G
                        35AB  1013
```

```
                        1015 HFLOAT_TO_HFLOA: $BAS$MAT_SC_MUL H, H
                        1016
                        1017
```

```
                                36C1  1019 ;+
                                36C1  1020 ; Multiply has been in byte.  Determine destination type to convert to dest.
                                36C1  1021 ;-
                                36C1  1022
                                36C1  1023 DEST_CASE_B:
                          56    5A    D0  36C1  1024          MOVL    R10, R6                         ; save original pointer
                05    06    02 A6    8F  36C4  1025 31$:      CASEB   DSC$B_DTYPE(R6), #DSC$K_DTYPE_B, #<DSC$K_DTYPE_D - DSC$K_DTYPE_B>
                                018D' 36C9  1026 1$:       .WORD   STORE_BYTE-1$                   ; no conversion needed
                                026D' 36CB  1027           .WORD   DEST_B_TO_W-1$                  ; code for word dtype
                                036F' 36CD  1028           .WORD   DEST_B_TO_L-1$                  ; code for long dtype
                                C9E5  36CF  1029           .WORD   ERR_DATTYPERR-1$                ; quad not supported
                                0471' 36D1  1030           .WORD   DEST_B_TO_F-1$                  ; code for float dtype
                                0573' 36D3  1031           .WORD   DEST_B_TO_D-1$                  ; code for double dtype
                                      36D5  1032
                                      36D5  1033 ;+
                                      36D5  1034 ; Check for g and h floating separately, since they fall outside the range
                                      36D5  1035 ; of the CASEB statement.
                                      36D5  1036 ;-
                                      36D5  1037
                      1B    02 A6    91  36D5  1038          CMPB    DSC$B_DTYPE(R6), #DSC$K_DTYPE_G
                                03    12  36D9  1039          BNEQ    2$
                                06BB  31  36DB  1040          BRW     DEST_B_TO_G
                                      36DE  1041
                      1C    02 A6    91  36DE  1042 2$:      CMPB    DSC$B_DTYPE(R6), #DSC$K_DTYPE_H
                                03    12  36E2  1043          BNEQ    3$
                                07CA  31  36E4  1044          BRW     DEST_B_TO_H
                                      36E7  1045
                      18    02 A6    91  36E7  1046 3$:      CMPB    DSC$B_DTYPE(R6), #DSC$K_DTYPE_DSC
                                06    12  36EB  1047          BNEQ    4$
                      56    04 A6    D0  36ED  1048          MOVL    4(R6), R6
                                D1    11  36F1  1049          BRB     31$
                                      36F3  1050
                                C9B8  31  36F3  1051 4$:      BRW     ERR_DATTYPERR                   ; unsupported dtype
                                      36F6  1052
                                      36F6  1053 ;+
                                      36F6  1054 ; Multiply has been in word.  Determine destination type to convert to dest.
                                      36F6  1055 ;-
                                      36F6  1056
                                      36F6  1057 DEST_CASE_W:
                          56    5A    D0  36F6  1058          MOVL    R10, R6
                05    06    02 A6    8F  36F9  1059 33$:      CASEB   DSC$B_DTYPE(R6), #DSC$K_DTYPE_B, #<DSC$K_DTYPE_D - DSC$K_DTYPE_B>
                                0136' 36FE  1060 1$:       .WORD   DEST_W_TO_B-1$                  ; code for byte dtype
                                025A' 3700  1061           .WORD   STORE_WORD-1$                   ; no conversion needed
                                033F' 3702  1062           .WORD   DEST_W_TO_L-1$                  ; code for long dtype
                                C9B0  3704  1063           .WORD   ERR_DATTYPERR-1$                ; quad not supported
                                0441' 3706  1064           .WORD   DEST_W_TO_F-1$                  ; code for float dtype
                                0547' 3708  1065           .WORD   DEST_W_TO_D-1$                  ; code for double dtype
                                      370A  1066
                                      370A  1067 ;+
                                      370A  1068 ; Check for g and h floating separately, since they fall outside the range
                                      370A  1069 ; of the CASEB statement.
                                      370A  1070 ;-
                                      370A  1071
                      1B    02 A6    91  370A  1072          CMPB    DSC$B_DTYPE(R6), #DSC$K_DTYPE_G
                                03    12  370E  1073          BNEQ    2$
                                068C  31  3710  1074          BRW     DEST_W_TO_G
                                      3713  1075
```

```
                 1C   02 A6   91   3713  1076 2$:      CMPB    DSC$B_DTYPE(R6), #DSC$K_DTYPE_H
                        03   12   3717  1077          BNEQ    3$
                      079B   31   3719  1078          BRW     DEST_W_TO_H
                             3710  1079
                 18   02 A6   91   3710  1080 3$:      CMPB    DSC$B_DTYPE(R6), #DSC$K_DTYPE_DSC
                        06   12   3720  1081          BNEQ    4$
                 56   04 A6   D0   3722  1082          MOVL    4(R6), R6
                        D1   11   3726  1083          BRB     33$
                             3728  1084
                      C983   31   3728  1085 4$:      BRW     ERR_DATTYPERR                   ; unsupported dtype
                             372B  1086
                             372B  1087 ;+
                             372B  1088 ; Multiply has been in long.  Determine destination type to convert to dest.
                             372B  1089 ;-
                             372B  1090
                             372B  1091 DEST_CASE_L:
                    56   5A   D0   372B  1092          MOVL    R10, R6
              05    06   02 A6   8F   372E  1093 35$:     CASEB   DSC$B_DTYPE(R6), #DSC$K_DTYPE_B, #<DSC$K_DTYPE_D - DSC$K_DTYPE_B>
                          0106' 3733  1094 1$:      .WORD   DEST_L_TO_B-1$                  ; code for byte dtype
                          0208' 3735  1095          .WORD   DEST_L_TO_W-1$                  ; code for word dtype
                          0327' 3737  1096          .WORD   STORE_LONG-1$                   ; no conversion needed
                          C97B  3739  1097          .WORD   ERR_DATTYPERR-1$                ; quad not supported
                          0411' 373B  1098          .WORD   DEST_L_TO_F-1$                  ; code for float dtype
                          051B' 373D  1099          .WORD   DEST_L_TO_D-1$                  ; code for double dtype
                             373F  1100
                             373F  1101 ;+
                             373F  1102 ; Check for g and h floating separately, since they fall outside the range
                             373F  1103 ; of the CASEB statement.
                             373F  1104 ;-
                             373F  1105
                 1B   02 A6   91   373F  1106          CMPB    DSC$B_DTYPE(R6), #DSC$K_DTYPE_G
                        03   12   3743  1107          BNEQ    2$
                      065D   31   3745  1108          BRW     DEST_L_TO_G
                             3748  1109
                 1C   02 A6   91   3748  1110 2$:      CMPB    DSC$B_DTYPE(R6), #DSC$K_DTYPE_H
                        03   12   374C  1111          BNEQ    3$
                      076C   31   374E  1112          BRW     DEST_L_TO_H
                             3751  1113
                 18   02 A6   91   3751  1114 3$:      CMPB    DSC$B_DTYPE(R6), #DSC$K_DTYPE_DSC
                        06   12   3755  1115          BNEQ    4$
                 56   04 A6   D0   3757  1116          MOVL    4(R6), R6
                        D1   11   375B  1117          BRB     35$
                             375D  1118
                      C94E   31   375D  1119 4$:      BRW     ERR_DATTYPERR                   ; unsupported dtype
                             3760  1120
                             3760  1121 ;+
                             3760  1122 ; Multiply has been in float.  Determine destination type to convert to dest.
                             3760  1123 ;-
                             3760  1124
                             3760  1125 DEST_CASE_F:
                    56   5A   D0   3760  1126          MOVL    R10, R6
              05    06   0C A6   8F   3763  1127 37$:     CASEB   DSC$B_DTYPE(R6), #DSC$K_DTYPE_B, #<DSC$K_DTYPE_D - DSC$K_DTYPE_B>
                          00D6' 3768  1128 1$:      .WORD   DEST_F_TO_B-1$                  ; code for byte dtype
                          01D8' 376A  1129          .WORD   DEST_F_TO_W-1$                  ; code for word dtype
                          02DA' 376C  1130          .WORD   DEST_F_TO_L-1$                  ; code for long dtype
                          C946  376E  1131          .WORD   ERR_DATTYPERR-1$                ; quad not supported
                          03F4' 3770  1132          .WORD   STORE_FLOAT-1$                  ; no conversion needed
```

```
                 04EF'  3772  1133          .WORD   DEST_F_TO_D-1$                  ; code for double dtype
                        3774  1134
                        3774  1135 ;+
                        3774  1136 ; Check for g and h floating separately, since they fall outside the range
                        3774  1137 ; of the CASEB statement.
                        3774  1138 ;-
                        3774  1139
          1B  02 A6  91 3774  1140          CMPB    DSC$B_DTYPE(R6), #DSC$K_DTYPE_G
                 03  12 3778  1141          BNEQ    2$
                062E  31 377A  1142          BRW     DEST_F_TO_G
                        377D  1143
          1C  02 A6  91 377D  1144 2$:      CMPB    DSC$B_DTYPE(R6), #DSC$K_DTYPE_H
                 03  12 3781  1145          BNEQ    3$
                073D  31 3783  1146          BRW     DEST_F_TO_H
                        3786  1147
          18  02 A6  91 3786  1148 3$:      CMPB    DSC$B_DTYPE(R6), #DSC$K_DTYPE_DSC
                 06  12 378A  1149          BNEQ    4$
          56  04 A6  D0 378C  1150          MOVL    4(R6), R6
                 D1  11 3790  1151          BRB     37$
                        3792  1152
                C919  31 3792  1153 4$:      BRW     ERR_DATTYPERR                   ; unsupported dtype
                        3795  1154
                        3795  1155 ;+
                        3795  1156 ; Multiply has been in double.
                        3795  1157 ; Determine destination type to convert to dest.
                        3795  1158 ;-
                        3795  1159
                        3795  1160 DEST_CASE_D:
             56  5A  D0 3795  1161          MOVL    R10, R6
       05 06  02 A6  8F 3798  1162 39$:     CASEB   DSC$B_DTYPE(R6), #DSC$K_DTYPE_B, #<DSC$K_DTYPE_D - DSC$K_DTYPE_B>
                 00A6' 379D  1163 1$:      .WORD   DEST_D_TO_B-1$                  ; code for byte dtype
                 01A8' 379F  1164          .WORD   DEST_D_TO_W-1$                  ; code for word dtype
                 02AA' 37A1  1165          .WORD   DEST_D_TO_L-1$                  ; code for long dtype
                 C911  37A3  1166          .WORD   ERR_DATTYPERR-1$                ; quad not supported
                 03AC' 37A5  1167          .WORD   DEST_D_TO_F-1$                  ; code for float dtype
                 051C' 37A7  1168          .WORD   STORE_DOUBLE-1$                 ; store it as is
                        37A9  1169
                        37A9  1170 ;+
                        37A9  1171 ; Check for g and h floating separately, since they fall outside the range
                        37A9  1172 ; of the CASEB statement.
                        37A9  1173 ;-
                        37A9  1174
          1B  02 A6  91 37A9  1175          CMPB    DSC$B_DTYPE(R6), #DSC$K_DTYPE_G
                 03  12 37AD  1176          BNEQ    2$
                05FF  31 37AF  1177          BRW     DEST_D_TO_G
                        37B2  1178
          1C  02 A6  91 37B2  1179 2$:      CMPB    DSC$B_DTYPE(R6), #DSC$K_DTYPE_H
                 03  12 37B6  1180          BNEQ    3$
                070E  31 37B8  1181          BRW     DEST_D_TO_H
                        37BB  1182
          18  02 A6  91 37BB  1183 3$:      CMPB    DSC$B_DTYPE(R6), #DSC$K_DTYPE_DSC
                 06  12 37BF  1184          BNEQ    4$
          56  04 A6  D0 37C1  1185          MOVL    4(R6), R6
                 D1  11 37C5  1186          BRB     39$
                        37C7  1187
                C8E4  31 37C7  1188 4$:      BRW     ERR_DATTYPERR                   ; unsupported dtype
                        37CA  1189
```

```
                         37CA  1190 ;+
                         37CA  1191 ; Multiply has been in gfloat.  Determine destination type to convert to dest.
                         37CA  1192 ;-
                         37CA  1193
                         37CA  1194 DEST_CASE_G:
          56    5A   D0  37CA  1195          MOVL    R10, R6
05    06    02 A6    8F  37CD  1196 41$:     CASEB   DSC$B_DTYPE(R6), #DSC$K_DTYPE_B, #<DSC$K_DTYPE_D - DSC$K_DTYPE_B>
               007A'     37D2  1197 1$:      .WORD   DEST_G_TO_B-1$                        ; code for byte dtype
               017C'     37D4  1198          .WORD   DEST_G_TO_W-1$                        ; code for word dtype
               027E'     37D6  1199          .WORD   DEST_G_TO_L-1$                        ; code for long dtype
               C8DC      37D8  1200          .WORD   ERR_DATTYPERR-1$                      ; quad not supported
               0380'     37DA  1201          .WORD   DEST_G_TO_F-1$                        ; code for float dtype
               04A0'     37DC  1202          .WORD   DEST_G_TO_D-1$                        ; code for double dtype
                         37DE  1203
                         37DE  1204 ;+
                         37DE  1205 ; Check for g and h floating separately, since they fall outside the range
                         37DE  1206 ; of the CASEB statement.
                         37DE  1207 ;-
                         37DE  1208
      1B    02 A6    91  37DE  1209          CMPB    DSC$B_DTYPE(R6), #DSC$K_DTYPE_G
               03    12  37E2  1210          BNEQ    2$
             05E5    31  37E4  1211          BRW     STORE_GFLOAT
                         37E7  1212
      1C    02 A6    91  37E7  1213 2$:      CMPB    DSC$B_DTYPE(R6), #DSC$K_DTYPE_H
               03    12  37EB  1214          BNEQ    3$
             06E3    31  37ED  1215          BRW     DEST_G_TO_H
                         37F0  1216
      18    02 A6    91  37F0  1217 3$:      CMPB    DSC$B_DTYPE(R6), #DSC$K_DTYPE_DSC
               06    12  37F4  1218          BNEQ    4$
      56    04 A6    D0  37F6  1219          MOVL    4(R6), R6
               D1    11  37FA  1220          BRB     41$
                         37FC  1221
             C8AF    31  37FC  1222 4$:      BRW     ERR_DATTYPERR                         ; unsupported dtype
                         37FF  1223
                         37FF  1224 ;+
                         37FF  1225 ; Multiply has been in hfloat.  Determine destination type to convert to dest.
                         37FF  1226 ;-
                         37FF  1227
                         37FF  1228 DEST_CASE_H:
          56    5A   D0  37FF  1229          MOVL    R10, R6
05    06    02 A6    8F  3802  1230 43$:     CASEB   DSC$B_DTYPE(R6), #DSC$K_DTYPE_B, #<DSC$K_DTYPE_D - DSC$K_DTYPE_B>
               004B'     3807  1231 1$:      .WORD   DEST_H_TO_B-1$                        ; code for byte dtype
               014D'     3809  1232          .WORD   DEST_H_TO_W-1$                        ; code for word dtype
               024F'     380B  1233          .WORD   DEST_H_TO_L-1$                        ; code for long dtype
               C8A7      380D  1234          .WORD   ERR_DATTYPERR-1$                      ; quad not supported
               0351'     380F  1235          .WORD   DEST_H_TO_F-1$                        ; code for float dtype
               0498'     3811  1236          .WORD   DEST_H_TO_D-1$                        ; code for double dtype
                         3813  1237
                         3813  1238 ;+
                         3813  1239 ; Check for g and h floating separately, since they fall outside the range
                         3813  1240 ; of the CASEB statement.
                         3813  1241 ;-
                         3813  1242
      1B    02 A6    91  3813  1243          CMPB    DSC$B_DTYPE(R6), #DSC$K_DTYPE_G
               03    12  3817  1244          BNEQ    2$
             05AC    31  3819  1245          BRW     DEST_H_TO_G
                         381C  1246
```

```
1C    02 A6    91  381C  2$:       CMPB    DSC$B_DTYPE(R6), #DSC$K_DTYPE_H
         03    12  3820            BNEQ    3$
       06B2    31  3822            BRW     STORE_HFLOAT
                   3825
18    02 A6    91  3825  3$:       CMPB    DSC$B_DTYPE(R6), #DSC$K_DTYPE_DSC
         06    12  3829            BNEQ    4$
56    04 A6    D0  382B            MOVL    4(R6), R6
         D1    11  382F            BRB     43$
                   3831
       C87A    31  3831  4$:       BRW     ERR_DATTYPERR                   ; unsupported dtype
                   3834
                   3834  DEST_W_TO_B:
   50    50    33  3834            CVTWB   R0, R0                          ; convert
         1D    11  3837            BRB     STORE_BYTE                      ; go store
                   3839
                   3839  DEST_L_TO_B:
   50    50    F6  3839            CVTLB   R0, R0                          ; convert
         18    11  383C            BRB     STORE_BYTE                      ; go store
                   383E
                   383E  DEST_F_TO_B:
   50    50    48  383E            CVTFB   R0, R0                          ; convert
         13    11  3841            BRB     STORE_BYTE                      ; go store
                   3843
                   3843  DEST_D_TO_B:
50    1C A4    66  3843            DIVD2   scale(R4), R0                   ; descale the double
   50    50    68  3847            CVTDB   R0, R0                          ; convert
         0A    11  384A            BRB     STORE_BYTE                      ; go store
                   384C
                   384C  DEST_G_TO_B:
   50    50 48FD   384C            CVTGB   R0, R0                          ; convert
         04    11  3850            BRB     STORE_BYTE                      ; go store
                   3852
                   3852  DEST_H_TO_B:
   50    50 68FD   3852            CVTHB   R0, R0                          ; convert
                   3856                                                    ; fall into store
                   3856
                   3856  STORE_BYTE:
   51    5A    D0  3856            MOVL    R10, R1                         ; pointer to dest descriptor
52    04 A4    D0  3859            MOVL    lower_bnd1(R4), R2              ; current row
   53    5B    D0  385D            MOVL    R11, R3                         ; current column
                   3860  ;+
                   3860  ; Redefine the following offsets for the call to the STORE macro.  The
                   3860  ; BSBW to here added 4 to the stack.
                   3860  ;-
                   3860
        0000002C   3860  value_desc = 44
        0000002C   3860  str_len = 44
        0000002E   3860  dtype = 46
        0000002F   3860  class = 47
        00000030   3860  pointer = 48
        00000034   3860  data = 52
                   3860
34 AE    50    90  3860            MOVB    R0, data(SP)
                   3864            STORE   B                               ; store
                   3935  ;+
                   3935  ; Restore the following offsets.
                   3935  ;-
```

```
                     3935  1304
                     3935  1305 value_desc = 40
                     3935  1306 str_len = 40
                     3935  1307 dtype = 42
                     3935  1308 class = 43
                     3935  1309 pointer = 44
                     3935  1310 data = 48
                     3935  1311
                05   3935  1312          RSB                                     ; go continue loop
                     3936  1313
                     3936  1314 DEST_B_TO_W:
     50   50   99   3936  1315          CVTBW   R0, R0                          ; convert
          1D   11   3939  1316          BRB     STORE_WORD                      ; go store
                     393B  1317
                     393B  1318 DEST_L_TO_W:
     50   50   F7   393B  1319          CVTLW   R0, R0                          ; convert
          18   11   393E  1320          BRB     STORE_WORD                      ; go store
                     3940  1321
                     3940  1322 DEST_F_TO_W:
     50   50   49   3940  1323          CVTFW   R0, R0                          ; convert
          13   11   3943  1324          BRB     STORE_WORD                      ; go store
                     3945  1325
                     3945  1326 DEST_D_TO_W:
50   1C A4   66   3945  1327          DIVD2   scale(R4), R0                   ; descale the double
     50   50   69   3949  1328          CVTDW   R0, R0                          ; convert to word
          0A   11   394C  1329          BRB     STORE_WORD                      ; go store
                     394E  1330
                     394E  1331 DEST_G_TO_W:
     50   50 49FD   394E  1332          CVTGW   R0, R0                          ; convert
          04   11   3952  1333          BRB     STORE_WORD                      ; go store
                     3954  1334
                     3954  1335 DEST_H_TO_W:
   50   50 69FD   3954  1336          CVTHW   R0, R0                          ; convert
                     3958  1337                                                  ; fall into store
                     3958  1338
                     3958  1339 STORE_WORD:
   51   5A   D0   3958  1340          MOVL    R10, R1                         ; pointer to dest descriptor
52   04 A4   D0   395B  1341          MOVL    lower_bnd1(R4), R2              ; current row
   53   5B   D0   395F  1342          MOVL    R11, R3                         ; current column
                     3962  1343 ;+
                     3962  1344 ; Redefine the following offsets for the call to the STORE macro.  The
                     3962  1345 ; BSBW to here added 4 to the stack.
                     3962  1346 ;-
                     3962  1347
                00000030   3962  1348 value_desc = 44
                0000002C   3962  1349 str_len = 44
                0000002E   3962  1350 dtype = 46
                0000002F   3962  1351 class = 47
                00000030   3962  1352 pointer = 48
                00000034   3962  1353 data = 52
                     3962  1354
34 AE   50   B0   3962  1355          MOVW    R0, data(SP)
                     3966  1356          STORE   W                               ; store
                     3A37  1357 ;+
                     3A37  1358 ; Restore the following offsets.
                     3A37  1359 ;-
                     3A37  1360
```

```
                00000028   3A37  1361 value_desc = 40
                00000028   3A37  1362 str_len = 40
                0000002A   3A37  1363 dtype = 42
                0000002B   3A37  1364 class = 43
                0000002C   3A37  1365 pointer = 44
                00000030   3A37  1366 data = 48
                     05    3A37  1367         RSB                                 ; go continue loop
                           3A38  1368
                           3A38  1369 DEST_B_TO_L:
          50    50   98    3A38  1370         CVTBL   R0, R0                      ; convert
                1D   11    3A3B  1371         BRB     STORE_LONG                  ; go store
                           3A3D  1372
                           3A3D  1373 DEST_W_TO_L:
          50    50   32    3A3D  1374         CVTWL   R0, R0                      ; convert
                18   11    3A40  1375         BRB     STORE_LONG                  ; go store
                           3A42  1376
                           3A42  1377 DEST_F_TO_L:
          50    50   4A    3A42  1378         CVTFL   R0, R0                      ; convert
                13   11    3A45  1379         BRB     STORE_LONG                  ; go store
                           3A47  1380
                           3A47  1381 DEST_D_TO_L:
    50    1C A4    66    3A47  1382         DIVD2   scale(R4), R0               ; descale the double
          50   50   6A    3A4B  1383         CVTDL   R0, R0                      ; convert
                0A   11    3A4E  1384         BRB     STORE_LONG                  ; go store
                           3A50  1385
                           3A50  1386 DEST_G_TO_L:
          50    50 4AFD    3A50  1387         CVTGL   R0, R0                      ; convert
                04   11    3A54  1388         BRB     STORE_LONG                  ; go store
                           3A56  1389
                           3A56  1390 DEST_H_TO_L:
          50    50 6AFD    3A56  1391         CVTHL   R0, R0                      ; convert
                           3A5A  1392                                             ; fall into store
                           3A5A  1393
                           3A5A  1394 STORE_LONG:
          51    5A   D0    3A5A  1395         MOVL    R10, R1                     ; pointer to dest descriptor
    52    04 A4    D0    3A5D  1396         MOVL    lower_bnd1(R4), R2          ; current row
          53    5B   D0    3A61  1397         MOVL    R11, R3                     ; current column
                           3A64  1398 ;+
                           3A64  1399 ; Redefine the following offsets for the call to the STORE macro.  The
                           3A64  1400 ; BSBW to here added 4 to the stack.
                           3A64  1401 ;-
                           3A64  1402
                0000002C   3A64  1403 value_desc = 44
                0000002C   3A64  1404 str_len = 44
                0000002E   3A64  1405 dtype = 46
                0000002F   3A64  1406 class = 47
                00000030   3A64  1407 pointer = 48
                00000034   3A64  1408 data = 52
                           3A64  1409
    34 AE         50   D0    3A64  1410         MOVL    R0, data(SP)
                           3A68  1411         STORE   L                           ; store
                           3B39  1412 ;+
                           3B39  1413 ; Restore the following offsets.
                           3B39  1414 ;-
                           3B39  1415
                00000028   3B39  1416 value_desc = 40
                00000028   3B39  1417 str_len = 40
```

```
                0000002A  3B39  1418 dtype = 42
                0000002B  3B39  1419 class = 43
                0000002C  3B39  1420 pointer = 44
                00000030  3B39  1421 data = 48
                      05  3B39  1422         RSB                                 ; go continue loop
                          3B3A  1423
                          3B3A  1424 DEST_B_TO_F:
          50    50    4C  3B3A  1425         CVTBF   R0, R0                      ; convert
                1D    11  3B3D  1426         BPB     STORE_FLOAT                 ; go store
                          3B3F  1427
                          3B3F  1428 DEST_W_TO_F:
          50    50    4D  3B3F  1429         CVTWF   R0, R0                      ; convert
                18    11  3B42  1430         BRB     STORE_FLOAT                 ; go store
                          3B44  1431
                          3B44  1432 DEST_L_TO_F:
          50    50    4E  3B44  1433         CVTLF   R0, R0                      ; convert
                13    11  3B47  1434         BRB     STORE_FLOAT                 ; go store
                          3B49  1435
                          3B49  1436 DEST_D_TO_F:
    50    1C A4     66  3B49  1437         DIVD2   scale(R4), R0               ; descale the double
          50    50  76  3B4D  1438         CVTDF   R0, R0                      ; convert
                0A    11  3B50  1439         BRB     STORE_FLOAT                 ; go store
                          3B52  1440
                          3B52  1441 DEST_G_TO_F:
          50    50 33FD  3B52  1442         CVTGF   R0, R0                      ; convert
                04    11  3B56  1443         BRB     STORE_FLOAT                 ; go store
                          3B58  1444
                          3B58  1445 DEST_H_TO_F:
          50    50 F6FD  3B58  1446         CVTHF   R0, R0                      ; convert
                          3B5C  1447                                             ; fall into store
                          3B5C  1448
                          3B5C  1449 STORE_FLOAT:
       51    5A    D0   3B5C  1450         MOVL    R10, R1                     ; pointer to dest descriptor
    52    04 A4    D0   3B5F  1451         MOVL    lower_bnd1(R4), R2          ; current row
       53    5B    D0   3B63  1452         MOVL    R11, R3                     ; current column
                          3B66  1453 ;+
                          3B66  1454 ; Redefine the following offsets for the call to the STORE macro.  The
                          3B66  1455 ; BSBW to here added 4 to the stack.
                          3B66  1456 ;-
                          3B66  1457
                0000002C  3B66  1458 value_desc = 44
                0000002C  3B66  1459 str_len = 44
                0000002E  3B66  1460 dtype = 46
                0000002F  3B66  1461 class = 47
                00000030  3B66  1462 pointer = 48
                00000034  3B66  1463 data = 52
                          3B66  1464
 34 AE    50    50  3B66  1465         MOVF    R0, data(SP)
                          3B6A  1466         STORE   F                           ; store
                          3C3B  1467 ;+
                          3C3B  1468 ; Restore the following offsets.
                          3C3B  1469 ;-
                          3C3B  1470
                00000028  3C3B  1471 value_desc = 40
                00000028  3C3B  1472 str_len = 40
                0000002A  3C3B  1473 dtype = 42
                0000002B  3C3B  1474 class = 43
```

```
                0000002C  3C3B  1475 pointer = 44
                00000030  3C3B  1476 data = 48
                      05  3C3B  1477          RSB                                 ; go continue loop
                          3C3C  1478                                              ; fall into store float
                          3C3C  1479 DEST_B_TO_D:
          50    50   6C   3C3C  1480          CVTBD   R0, R0                      ; get double
    50    1C A4    64   3C3F  1481          MULD2   scale(R4), R0               ; scale the double
                          3C43  1482                                              ;  NOTE: no need for integerize
                74    11  3C43  1483          BRB     STORE_DOUBLE                ; go store
                          3C45  1484
                          3C45  1485 DEST_W_TO_D:
          50    50   6D   3C45  1486          CVTWD   R0, R0                      ; get double
    50    1C A4    64   3C48  1487          MULD2   scale(R4), R0               ; scale the double
                          3C4C  1488                                              ;  NOTE: no need for integerize
                6B    11  3C4C  1489          BRB     STORE_DOUBLE                ; go store
                          3C4E  1490
                          3C4E  1491 DEST_L_TO_D:
          50    50   6E   3C4E  1492          CVTLD   R0, R0                      ; get double
    50    1C A4    64   3C51  1493          MULD2   scale(R4), R0               ; scale for dest
                          3C55  1494                                              ;  NOTE: no need for integerize
                62    11  3C55  1495          BRB     STORE_DOUBLE                ;  go store
                          3C57  1496
                          3C57  1497 DEST_F_TO_D:
          50    50   56   3C57  1498          CVTFD   R0, R0                      ; get double
    50    1C A4    64   3C5A  1499          MULD2   scale(R4), R0               ; scale for dest
    08    1C A4    71   3C5E  1500          CMPD    scale(R4), #1               ; scale 0?
                0C    13  3C62  1501          BEQL    1$                          ; yes, don't integerize
          7E    54   D0   3C64  1502          MOVL    R4, -(SP)                   ; save R4
00000000'GF     16  3C67  1503          JSB     G^MTH$DINT_R4               ; integerize
          54    8E   D0   3C6D  1504          MOVL    (SP)+, R4                   ; restore R4
                47    11  3C70  1505 1$:      BRB     STORE_DOUBLE                ; go store
                          3C72  1506
                          3C72  1507 DEST_G_TO_D:
                          3C72  1508          ;+
                          3C72  1509          ; Note the intermediate conversion to hfloat.
                          3C72  1510          ;-
          7E    52   D0   3C72  1511          MOVL    R2, -(SP)                   ; save regs which CVTGH
          7E    53   D0   3C75  1512          MOVL    R3, -(SP)                   ;  will destroy
          50    50 56FD   3C78  1513          CVTGH   R0, R0                      ; cvt gfloat to hfloat
          50    50 F7FD   3C7C  1514          CVTHD   R0, R0                      ; cvt to desired double
          53    8E   D0   3C80  1515          MOVL    (SP)+, R3                   ; restore regs
          52    8E   D0   3C83  1516          MOVL    (SP)+, R2
    50    1C A4    64   3C86  1517          MULD2   scale(R4), R0               ; scale for dest
    08    1C A4    71   3C8A  1518          CMPD    scale(R4), #1               ; scale 0?
                29    13  3C8E  1519          BEQL    STORE_DOUBLE                ; yes, don't integerize
          7E    54   D0   3C90  1520          MOVL    R4, -(SP)                   ; save R4
00000000'GF     16  3C93  1521          JSB     G^MTH$DINT_R4               ; integerize
          54    8E   D0   3C99  1522          MOVL    (SP)+, R4                   ; restore R4
                001A  31  3C9C  1523          BRW     STORE_DOUBLE
                          3C9F  1524
                          3C9F  1525 DEST_H_TO_D:
          50    50 F7FD   3C9F  1526          CVTHD   R0, R0                      ; get double
    50    1C A4    64   3CA3  1527          MULD2   scale(R4), R0               ; scale for dest
    08    1C A4    71   3CA7  1528          CMPD    scale(R4), #1               ; scale 0?
                0C    13  3CAB  1529          BEQL    STORE_DOUBLE                ; yes, don't integerize
          7E    54   D0   3CAD  1530          MOVL    R4, -(SP)                   ; save R4
00000000'GF     16  3CB0  1531          JSB     G^MTH$DINT_R4               ; integerize
```

```
        54    8E    D0 3CB6          MOVL    (SP)+, R4                    ; restore R4
                       3CB9                                               ; fall into store
                       3CB9
                       3CB9 STORE_DOUBLE:
        52    5A    D0 3CB9          MOVL    R10, R2                      ; pointer to dest descriptor
     53    04 A4    D0 3CBC          MOVL    lower_bnd1(R4), R3           ; current row
        54    5B    D0 3CC0          MOVL    R11, R4                      ; current column
                       3CC3 ;+
                       3CC3 ; Redefine the following offsets for the call to the STORE macro.  The
                       3CC3 ; BSBW to here added 4 to the stack.
                       3CC3 ;-
                       3CC3
           0000002C    3CC3 value_desc = 44
           0000002C    3CC3 str_len = 44
           0000002E    3CC3 dtype = 46
           0000002F    3CC3 class = 47
           00000030    3CC3 pointer = 48
           00000034    3CC3 data = 52
                       3CC3
     34 AE    50    70 3CC3          MOVD    R0, data(SP)
                       3CC7          STORE   D                            ; store
                       3D98 ;+
                       3D98 ; Restore the following offsets.
                       3D98 ;-
                       3D98
           00000028    3D98 value_desc = 40
           00000028    3D98 str_len = 40
           0000002A    3D98 dtype = 42
           0000002B    3D98 class = 43
           0000002C    3D98 pointer = 44
           00000030    3D98 data = 48
                 05    3D98          RSB                                  ; go continue loop
                       3D99
                       3D99 DEST_B_TO_G:
        50    50 4CFD  3D99          CVTBG   R0, R0                       ; convert
              2D    11 3D9D          BRB     STORE_GFLOAT                 ; go store
                       3D9F
                       3D9F DEST_W_TO_G:
        50    50 4DFD  3D9F          CVTWG   R0, R0                       ; convert
              27    11 3DA3          BRB     STORE_GFLOAT                 ; go store
                       3DA5
                       3DA5 DEST_L_TO_G:
        50    50 4EFD  3DA5          CVTLG   R0, R0                       ; convert
              21    11 3DA9          BRB     STORE_GFLOAT                 ; go store
                       3DAB
                       3DAB DEST_F_TO_G:
        50    50 99FD  3DAB          CVTFG   R0, R0                       ; convert
              1B    11 3DAF          BRB     STORE_GFLOAT                 ; go store
                       3DB1
                       3DB1 DEST_D_TO_G:
                       3DB1          ;+
                       3DB1          ; Note the intermediate conversion to hfloat.
                       3DB1          ;-
        7E    52    D0 3DB1          MOVL    R2, -(SP)                    ; save regs which CVTDH
        7E    53    D0 3DB4          MOVL    R3, -(SP)                    ;  will destroy
        50    50 32FD  3DB7          CVTDH   R0, R0                       ; cvt dbl to hfloat
        50    50 76FD  3DBB          CVTHG   R0, R0                       ; cvt to desired gfloat
```

```
                53   8E    D0  3DBF  1589          MOVL    (SP)+, R3                       ; restore regs
                52   8E    D0  3DC2  1590          MOVL    (SP)+, R2
                   0004    31  3DC5  1591          BRW     STORE_GFLOAT
                               3DC8  1592
                               3DC8  1593 DEST_H_TO_G:
                50   50 76FD  3DC8  1594          CVTHG   R0, R0                          ; convert
                               3DCC  1595                                                  ; fall into store
                               3DCC  1596
                               3DCC  1597 STORE_GFLOAT:
                52   5A    D0  3DCC  1598          MOVL    R10, R2                         ; pointer to dest descriptor
            53   04 A4    D0  3DCF  1599          MOVL    lower_bnd1(R4), R3              ; current row
                54   5B    D0  3DD3  1600          MOVL    R11, R4                         ; current column
                               3DD6  1601 ;+
                               3DD6  1602 ; Redefine the following offsets for the call to the STORE macro.  The
                               3DD6  1603 ; BSBW to here added 4 to the stack.
                               3DD6  1604 ;-
                               3DD6  1605
                     0000002C  3DD6  1606 value_desc = 44
                     0000002C  3DD6  1607 str_len = 44
                     0000002E  3DD6  1608 dtype = 46
                     0000002F  3DD6  1609 class = 47
                     00000030  3DD6  1610 pointer = 48
                     00000034  3DD6  1611 data = 52
                               3DD6  1612
            34 AE    50 50FD  3DD6  1613          MOVG    R0, data(SP)
                               3DDB  1614          STORE   G                               ; store
                               3EB0  1615 ;+
                               3EB0  1616 ; Restore the following offsets.
                               3EB0  1617 ;-
                               3EB0  1618
                     00000028  3EB0  1619 value_desc = 40
                     00000028  3EB0  1620 str_len = 40
                     0000002A  3EB0  1621 dtype = 42
                     0000002B  3EB0  1622 class = 43
                     0000002C  3EB0  1623 pointer = 44
                     00000030  3EB0  1624 data = 48
                           05  3EB0  1625          RSB                                     ; go continue loop
                               3EB1  1626
                               3EB1  1627 DEST_B_TO_H:
                50   50 6CFD  3EB1  1628          CVTBH   R0, R0                          ; convert
                     20   11  3EB5  1629          BRB     STORE_HFLOAT                    ; go store
                               3EB7  1630
                               3EB7  1631 DEST_W_TO_H:
                50   50 6DFD  3EB7  1632          CVTWH   R0, R0                          ; convert
                     1A   11  3EBB  1633          BRB     STORE_HFLOAT                    ; go store
                               3EBD  1634
                               3EBD  1635 DEST_L_TO_H:
                50   50 6EFD  3EBD  1636          CVTLH   R0, R0                          ; convert
                     14   11  3EC1  1637          BRB     STORE_HFLOAT                    ; go store
                               3EC3  1638
                               3EC3  1639 DEST_F_TO_H:
                50   50 98FD  3EC3  1640          CVTFH   R0, R0                          ; convert
                     0E   11  3EC7  1641          BRB     STORE_HFLOAT                    ; go store
                               3EC9  1642
                               3EC9  1643 DEST_D_TO_H:
            50   1C A4   66  3EC9  1644          DIVD2   scale(R4), R0                   ; descale the double
                50   50 32FD  3ECD  1645          CVTDH   R0, R0                          ; convert
```

```
                     04   11   3ED1  1646          BRB     STORE_HFLOAT                    ; go store
                               3ED3  1647
                               3ED3  1648 DEST_G_TO_H:
                50   50 56FD   3ED3  1649          CVTGH   R0, R0                          ; convert
                               3ED7  1650                                                  ; fall into store
                               3ED7  1651
                               3ED7  1652 STORE_HFLOAT:
             55   04 A4   D0   3ED7  1653          MOVL    lower_bnd1(R4), R5              ; current row
                54   5A   D0   3EDB  1654          MOVL    R10, R4                         ; pointer to dest descriptor
                56   5B   D0   3EDE  1655          MOVL    R11, R6                         ; current column
                               3EE1  1656 ;+
                               3EE1  1657 ; Redefine the following offsets for the call to the STORE macro.  The
                               3EE1  1658 ; BSBW to here added 4 to the stack.
                               3EE1  1659 ;-
                               3EE1  1660
                     0000002C  3EE1  1661 value_desc = 44
                     0000002C  3EE1  1662 str_len = 44
                     0000002E  3EE1  1663 dtype = 46
                     0000002F  3EE1  1664 class = 47
                     00000030  3EE1  1665 pointer = 48
                     00000034  3EE1  1666 data = 52
                               3EE1  1667
             34 AE   50 70FD   3EE1  1668          MOVH    R0, data(SP)
                               3EE6  1669          STORE   H                               ; store
                               3FBB  1670 ;+
                               3FBB  1671 ; Restore the following offsets.
                               3FBB  1672 ;-
                               3FBB  1673
                     00000028  3FBB  1674 value_desc = 40
                     00000028  3FBB  1675 str_len = 40
                     0000002A  3FBB  1676 dtype = 42
                     0000002B  3FBB  1677 class = 43
                     0000002C  3FBB  1678 pointer = 44
                     00000030  3FBB  1679 data = 48
                           05  3FBB  1680          RSB                                     ; go continue loop
                               3FBC  1681
                               3FBC  1682
                               3FBC  1683          .END                                    ; end of BAS$MAT_SCA_MUL
```

```
BAS$$SCALE_R1                   ********    X   00
BAS$$STOP                       ********    X   00
BAS$FETCH_BFA                   ********    X   00
BAS$FET_FA_B_R8                 ********    X   00
BAS$FET_FA_D_R8                 ********    X   00
BAS$FET_FA_F_R8                 ********    X   00
BAS$FET_FA_G_R8                 ********    X   00
BAS$FET_FA_H_R8                 ********    X   00
BAS$FET_FA_L_R8                 ********    X   00
BAS$FET_FA_W_R8                 ********    X   00
BAS$K_ARGDONMAT                 ********    X   00
BAS$K_DATTYPERR                 ********    X   00
BAS$MAT_REDIM                   ********    X   00
BAS$MAT_SCA_MUL                 00000000 RG     02
BAS$STORE_BFA                   ********    X   00
BAS$STO_FA_B_R8                 ********    X   00
BAS$STO_FA_D_R8                 ********    X   00
BAS$STO_FA_F_R8                 ********    X   00
BAS$STO_FA_G_R8                 ********    X   00
BAS$STO_FA_H_R8                 ********    X   00
BAS$STO_FA_L_R8                 ********    X   00
BAS$STO_FA_W_R8                 ********    X   00
BYTE                            000000BB R      02
BYTE_TO_BYTE                    000000F0 R      02
BYTE_TO_DOUBLE                  00000531 R      02
BYTE_TO_FLOAT                   00000420 R      02
BYTE_TO_GFLOAT                  00000642 R      02
BYTE_TO_HFLOAT                  00000756 R      02
BYTE_TO_LONG                    0000030F R      02
BYTE_TO_WORD                    000001FE R      02
DEST_B_TO_D                     00003C3C R      02
DEST_B_TO_F                     00003B3A R      02
DEST_B_TO_G                     00003D99 R      02
DEST_B_TO_H                     00003EB1 R      02
DEST_B_TO_L                     00003A38 R      02
DEST_B_TO_W                     00003936 R      02
DEST_CASE_B                     000036C1 R      02
DEST_CASE_D                     00003795 R      02
DEST_CASE_F                     00003760 R      02
DEST_CASE_G                     000037CA R      02
DEST_CASE_H                     000037FF R      02
DEST_CASE_L                     0000372B R      02
DEST_CASE_W                     000036F6 R      02
DEST_D_TO_B                     00003843 R      02
DEST_D_TO_F                     00003B49 R      02
DEST_D_TO_G                     00003DB1 R      02
DEST_D_TO_H                     00003EC9 R      02
DEST_D_TO_L                     00003A47 R      02
DEST_D_TO_W                     00003945 R      02
DEST_F_TO_B                     0000383E R      02
DEST_F_TO_D                     00003C57 R      02
DEST_F_TO_G                     00003DAB R      02
DEST_F_TO_H                     00003EC3 R      02
DEST_F_TO_L                     00003A42 R      02
DEST_F_TO_W                     00003940 R      02
DEST_G_TO_B                     0000384C R      02
DEST_G_TO_D                     00003C72 R      02
DEST_G_TO_F                                             00003B52 R      02
DEST_G_TO_H                                             00003ED3 R      02
DEST_G_TO_L                                             00003A50 R      02
DEST_G_TO_W                                             0000394E R      02
DEST_H_TO_B                                             00003852 R      02
DEST_H_TO_D                                             00003C9F R      02
DEST_H_TO_F                                             00003B58 R      02
DEST_H_TO_G                                             00003DC8 R      02
DEST_H_TO_L                                             00003A56 R      02
DEST_H_TO_W                                             00003954 R      02
DEST_L_TO_B                                             00003839 R      02
DEST_L_TO_D                                             00003C4E R      02
DEST_L_TO_F                                             00003B44 R      02
DEST_L_TO_G                                             00003DA5 R      02
DEST_L_TO_H                                             00003EBD R      02
DEST_L_TO_W                                             0000393B R      02
DEST_MATRIX                                   =         00000010
DEST_W_TO_B                                             00003834 R      02
DEST_W_TO_D                                             00003C45 R      02
DEST_W_TO_F                                             00003B3F R      02
DEST_W_TO_G                                             00003D9F R      02
DEST_W_TO_H                                             00003EB7 R      02
DEST_W_TO_L                                             00003A3D R      02
DOUBLE                                                  00001F65 R      02
DOUBLE_TO_BYTE                                          00001F9A R      02
DOUBLE_TO_DOUBL                                         000023D2 R      02
DOUBLE_TO_FLOA                                          000022C4 R      02
DOUBLE_TO_GFLOA                                         000024F3 R      02
DOUBLE_TO_HFLOA                                         00002607 R      02
DOUBLE_TO_LONG                                          000021B6 R      02
DOUBLE_TO_WORD                                          000020A8 R      02
DSC$A_A0                                      =         00000010
DSC$B_AFLAGS                                  =         0000000A
DSC$B_CLASS                                   =         00000003
DSC$B_DIMCT                                   =         0000000B
DSC$B_DTYPE                                   =         00000002
DSC$K_CLASS_BFA                               =         000000BF
DSC$K_DTYPE_B                                 =         00000006
DSC$K_DTYPE_D                                 =         0000000B
DSC$K_DTYPE_DSC                               =         00000018
DSC$K_DTYPE_G                                 =         0000001B
DSC$K_DTYPE_H                                 =         0000001C
DSC$L_L1_1                                    =         00000018
DSC$L_L1_2                                    =         0000001C
DSC$L_L2_2                                    =         00000024
DSC$L_M1                                      =         00000014
DSC$L_M2                                      =         00000018
DSC$L_U1_1                                    =         0000001C
DSC$L_U1_2                                    =         00000020
DSC$L_U2_2                                    =         00000028
DSC$V_FL_BOUNDS                               =         00000007
DSC$W_LENGTH                                  =         00000000
ERR_ARGDONMAT                                           0000002E R      02
ERR_DATTYPERR                                           000000AE R      02
FLOAT                                                   000017BF R      02
FLOAT_TO_BYTE                                           000017F4 R      02
FLOAT_TO_DOUBL                                          00001C2C R      02
```

FLOAT_TO_FLOAT	00001B1E	R	02	LOOP_1ST_SUBGL	0000159D	R	02
FLOAT_TO_GFLOA	00001D3D	R	02	LOOP_1ST_SUBGW	00000DF4	R	02
FLOAT_TO_HFLOA	00001E51	R	02	LOOP_1ST_SUBHB	0000075C	R	02
FLOAT_TO_LONG	00001A10	R	02	LOOP_1ST_SUBHD	0000260D	R	02
FLOAT_TO_WORD	00001902	R	02	LOOP_1ST_SUBHF	00001E57	R	02
GFLOAT	0000271B	R	02	LOOP_1ST_SUBHG	00002DDE	R	02
GFLOAT_TO_BYTE	00002750	R	02	LOOP_1ST_SUBHH	000035B1	R	02
GFLOAT_TO_DOUBL	00002BA8	R	02	LOOP_1ST_SUBHL	000016B1	R	02
GFLOAT_TO_FLOA	00002A92	R	02	LOOP_1ST_SUBHW	00000F08	R	02
GFLOAT_TO_GFLOA	00002CC2	R	02	LOOP_1ST_SUBLB	00000314	R	02
GFLOAT_TO_HFLOA	00002DD8	R	02	LOOP_1ST_SUBLD	000021BB	R	02
GFLOAT_TO_LONG	0000297C	R	02	LOOP_1ST_SUBLF	00001A15	R	02
GFLOAT_TO_WORD	00002866	R	02	LOOP_1ST_SUBLG	00002982	R	02
HFLOAT	00002EF2	R	02	LOOP_1ST_SUBLH	00003159	R	02
HFLOAT_TO_BYTE	00002F27	R	02	LOOP_1ST_SUBLL	0000126C	R	02
HFLOAT_TO_DOUBL	0000337F	R	02	LOOP_1ST_SUBLW	00000AC0	R	02
HFLOAT_TO_FLOA	00003269	R	02	LOOP_1ST_SUBWB	00000203	R	02
HFLOAT_TO_GFLOA	00003495	R	02	LOOP_1ST_SUBWD	000020AD	R	02
HFLOAT_TO_HFLOA	000035AB	R	02	LOOP_1ST_SUBWF	00001907	R	02
HFLOAT_TO_LONG	00003153	R	02	LOOP_1ST_SUBWG	0000286C	R	02
HFLOAT_TO_WORD	0000303D	R	02	LOOP_1ST_SUBWH	00003043	R	02
INIT_ONE_SUB	0000003B	R	02	LOOP_1ST_SUBWL	0000115E	R	02
INIT_TWO_SUBS	00000059	R	02	LOOP_1ST_SUBWW	000009B2	R	02
LONG	00001016	R	02	LOOP_2ND_SUBBB	000000F8	R	02
LONG_TO_BYTE	0000104B	R	02	LOOP_2ND_SUBBD	00001FA2	R	02
LONG_TO_DOUBLE	00001486	R	02	LOOP_2ND_SUBBF	000017FC	R	02
LONG_TO_FLOAT	00001375	R	02	LOOP_2ND_SUBBG	00002759	R	02
LONG_TO_GFLOAT	00001597	R	02	LOOP_2ND_SUBBH	00002F30	R	02
LONG_TO_HFLOAT	000016AB	R	02	LOOP_2ND_SUBBL	00001053	R	02
LONG_TO_LONG	00001267	R	02	LOOP_2ND_SUBBW	000008A7	R	02
LONG_TO_WORD	00001159	R	02	LOOP_2ND_SUBDB	00000539	R	02
LOOP_1ST_SUBBB	000000F5	R	02	LOOP_2ND_SUBDD	000023DA	R	02
LOOP_1ST_SUBBD	00001F9F	R	02	LOOP_2ND_SUBDF	00001C34	R	02
LOOP_1ST_SUBBF	000017F9	R	02	LOOP_2ND_SUBDG	00002BB1	R	02
LOOP_1ST_SUBBG	00002756	R	02	LOOP_2ND_SUBDH	00003388	R	02
LOOP_1ST_SUBBH	00002F2D	R	02	LOOP_2ND_SUBDL	0000148E	R	02
LOOP_1ST_SUBBL	00001050	R	02	LOOP_2ND_SUBDW	00000CE5	R	02
LOOP_1ST_SUBBW	000008A4	R	02	LOOP_2ND_SUBFB	00000428	R	02
LOOP_1ST_SUBDB	00000536	R	02	LOOP_2ND_SUBFD	000022CC	R	02
LOOP_1ST_SUBDD	000023D7	R	02	LOOP_2ND_SUBFF	00001B26	R	02
LOOP_1ST_SUBDF	00001C31	R	02	LOOP_2ND_SUBFG	00002A9B	R	02
LOOP_1ST_SUBDG	00002BAE	R	02	LOOP_2ND_SUBFH	00003272	R	02
LOOP_1ST_SUBDH	00003385	R	02	LOOP_2ND_SUBFL	0000137D	R	02
LOOP_1ST_SUBDL	0000148B	R	02	LOOP_2ND_SUBFW	00000BD4	R	02
LOOP_1ST_SUBDW	00000CE2	R	02	LOOP_2ND_SUBGB	0000064B	R	02
LOOP_1ST_SUBFB	00000425	R	02	LOOP_2ND_SUBGD	000024FC	R	02
LOOP_1ST_SUBFD	000022C9	R	02	LOOP_2ND_SUBGF	00001D46	R	02
LOOP_1ST_SUBFF	00001B23	R	02	LOOP_2ND_SUBGG	00002CCB	R	02
LOOP_1ST_SUBFG	00002A98	R	02	LOOP_2ND_SUBGH	0000349E	R	02
LOOP_1ST_SUBFH	0000326F	R	02	LOOP_2ND_SUBGL	000015A0	R	02
LOOP_1ST_SUBFL	0000137A	R	02	LOOP_2ND_SUBGW	00000DF7	R	02
LOOP_1ST_SUBFW	00000BD1	R	02	LOOP_2ND_SUBHB	0000075F	R	02
LOOP_1ST_SUBGB	00000648	R	02	LOOP_2ND_SUBHD	00002610	R	02
LOOP_1ST_SUBGD	000024F9	R	02	LOOP_2ND_SUBHF	00001E5A	R	02
LOOP_1ST_SUBGF	00001D43	R	02	LOOP_2ND_SUBHG	00002DE1	R	02
LOOP_1ST_SUBGG	00002CC8	R	02	LOOP_2ND_SUBHH	000035B4	R	02
LOOP_1ST_SUBGH	0000349B	R	02	LOOP_2ND_SUBHL	000016B4	R	02

```
LOOP_2ND_SUBHW                  00000F0B R      02
LOOP_2ND_SUBLB                  00000317 R      02
LOOP_2ND_SUBLD                  000021BE R      02
LOOP_2ND_SUBLF                  00001A18 R      02
LOOP_2ND_SUBLG                  00002985 R      02
LOOP_2ND_SUBLH                  0000315C R      02
LOOP_2ND_SUBLL                  0000126F R      02
LOOP_2ND_SUBLW                  00000AC3 R      02
LOOP_2ND_SUBWB                  00000206 R      02
LOOP_2ND_SUBWD                  000020B0 R      02
LOOP_2ND_SUBWF                  0000190A R      02
LOOP_2ND_SUBWG                  0000286F R      02
LOOP_2ND_SUBWH                  00003046 R      02
LOOP_2ND_SUBWL                  00001161 R      02
LOOP_2ND_SUBWW                  000009B5 R      02
LOWER_BND1                    = 00000004
LOWER_BND2                    = 00000000
MTH$DINT_R4                     ********    X   00
SAVE_SCALAR                   = 0000000C
SCALAR                        = 00000004
SCALAR_DTYPE                  = 00000008
SCALE                         = 0000001C
SEPARATE_DTYPES                 0000007F R      02
SF$L_SAVE_FP                  = 0000000C
SRC_MATRIX                    = 0000000C
STORE_BYTE                      00003856 R      02
STORE_DOUBLE                    00003CB9 R      02
STORE_FLOAT                     00003B5C R      02
STORE_GFLOAT                    00003DCC R      02
STORE_HFLOAT                    00003ED7 R      02
STORE_LONG                      00003A5A R      02
STORE_WORD                      00003958 R      02
UPPER_BND1                    = 00000008
WORD                            0000086A R      02
WORD_TO_BYTE                    0000089F R      02
WORD_TO_DOUBLE                  00000CDD R      02
WORD_TO_FLOAT                   00000BCC R      02
WORD_TO_GFLOAT                  00000DEE R      02
WORD_TO_HFLOAT                  00000F02 R      02
WORD_TO_LONG                    00000ABB R      02
WORD_TO_WORD                    000009AD R      02
```

Psect synopsis

PSECT name	Allocation	PSECT No.	Attributes
. ABS .	00000000 (0.)	00 (0.)	NOPIC USR CON ABS LCL NOSHR NOEXE NORD NOWRT NOVEC BYTE
ABS	00000000 (0.)	01 (1.)	NOPIC USR CON ABS LCL NOSHR EXE RD WRT NOVEC BYTE
_BAS$CODE	00003FBC (16316.)	02 (2.)	PIC USR CON REL LCL SHR EXE RD NOWRT NOVEC LONG

Performance indicators

Phase	Page faults	CPU Time	Elapsed Time
Initialization	32	00:00:00.06	00:00:00.29
Command processing	132	00:00:00.53	00:00:02.61
Pass 1	672	00:00:32.81	00:01:09.86
Symbol table sort	0	00:00:01.45	00:00:03.15
Pass 2	308	00:00:09.08	00:00:21.43
Symbol table output	31	00:00:00.23	00:00:00.65
Psect synopsis output	2	00:00:00.02	00:00:00.02
Cross-reference output	0	00:00:00.00	00:00:00.00
Assembler run totals	1179	00:00:44.20	00:01:38.01

The working set limit was 2000 pages.
239417 bytes (468 pages) of virtual memory were used to buffer the intermediate code.
There were 50 pages of symbol table space allocated to hold 419 non-local and 573 local symbols.
1683 source lines were read in Pass 1, producing 54 object records in Pass 2.
40 pages of virtual memory were used to define 11 macros.

Macro library statistics

Macro library name	Macros defined
_255DUA28:[BASRTL.OBJ]BASRTL.MLB;1	2
_255DUA28:[SYSLIB]STARLET.MLB;2	5
TOTALS (all libraries)	7

493 GETS were required to define 7 macros.

There were no errors, warnings or information messages.

MACRO/ENABLE=SUPPRESSION/DISABLE=(GLOBAL,TRACEBACK)/LIS=LIS$:BASMATSCA/OBJ=OBJ$:BASMATSCA MSRC$:BASMATSCA/UPDATE=(ENH$:BASMATSCA)+LI

BASMATSUB
LIS

BASMATSCA
LIS

BASMATTRN
LIS