```
M 1

BBBBBBBBBBBB        AAAAAAAAA        SSSSSSSSSSSS  RRRRRRRRRRRR     TTTTTTTTTTTTTTT  LLL
BBBBBBBBBBBB        AAAAAAAAA        SSSSSSSSSSSS  RRRRRRRRRRRR     TTTTTTTTTTTTTTT  LLL
BBBBBBBBBBBB        AAAAAAAAA        SSSSSSSSSSSS  RRRRRRRRRRRR     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     RRRRRRRRRRRR           TTT        LLL
BBBBBBBBBBBB     AAA         AAA     SSSSSSSSS     RRRRRRRRRRRR           TTT        LLL
BBBBBBBBBBBB     AAA         AAA     SSSSSSSSS     RRRRRRRRRRRR           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  SSSSSSSSSSSS     RRR         RRR        TTT        LLLLLLLLLLLLLLL
BBBBBBBBBBBB     AAA         AAA  SSSSSSSSSSSS     RRR         RRR        TTT        LLLLLLLLLLLLLLL
BBBBBBBBBBBB     AAA         AAA  SSSSSSSSSSSS     RRR         RRR        TTT        LLLLLLLLLLLLLLL
```

```
**FILE**ID**BASMATMUL                              D 9
```

```
BBBBBBBB      AAAAAA      SSSSSSSS  MM      MM    AAAAAA    TTTTTTTTTT  MM      MM  UU      UU  LL
BBBBBBBB      AAAAAA      SSSSSSSS  MM      MM    AAAAAA    TTTTTTTTTT  MM      MM  UU      UU  LL
BB      BB  AA      AA  SS          MMMM  MMMM  AA      AA      TT      MMMM  MMMM  UU      UU  LL
BB      BB  AA      AA  SS          MMMM  MMMM  AA      AA      TT      MMMM  MMMM  UU      UU  LL
BB      BB  AA      AA  SS          MM  MM  MM  AA      AA      TT      MM  MM  MM  UU      UU  LL
BB      BB  AA      AA  SS          MM  MM  MM  AA      AA      TT      MM  MM  MM  UU      UU  LL
BBBBBBBB    AA      AA    SSSSSS    MM      MM  AA      AA      TT      MM      MM  UU      UU  LL
BBBBBBBB    AA      AA    SSSSSS    MM      MM  AA      AA      TT      MM      MM  UU      UU  LL
BB      BB  AAAAAAAAAA          SS  MM      MM  AAAAAAAAAA      TT      MM      MM  UU      UU  LL
BB      BB  AAAAAAAAAA          SS  MM      MM  AAAAAAAAAA      TT      MM      MM  UU      UU  LL
BB      BB  AA      AA          SS  MM      MM  AA      AA      TT      MM      MM  UU      UU  LL              ....
BB      BB  AA      AA          SS  MM      MM  AA      AA      TT      MM      MM  UU      UU  LL              ....
BBBBBBBB    AA      AA  SSSSSSSS    MM      MM  AA      AA      TT      MM      MM  UUUUUUUUUU  LLLLLLLLLL      ....
BBBBBBBB    AA      AA  SSSSSSSS    MM      MM  AA      AA      TT      MM      MM  UUUUUUUUUU  LLLLLLLLLL      ....

LL              IIIIII      SSSSSSSS
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            .TITLE  BAS$MAT_MUL                     ; BASIC matrix multiply
0000            .IDENT  /1-021/                         ; File: BASMATMUL.MAR   Edit: SBL1020
0000
0000      ;
0000      ;****************************************************************************
0000      ;*                                                                          *
0000      ;*  COPYRIGHT (c) 1978, 1980, 1982, 1984 BY                                 *
0000      ;*  DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.                  *
0000      ;*  ALL RIGHTS RESERVED.                                                    *
0000      ;*                                                                          *
0000      ;*  THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED   *
0000      ;*  ONLY IN  ACCORDANCE WITH  THE  TERMS  OF  SUCH  LICENSE  AND WITH THE   *
0000      ;*  INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR  ANY  OTHER   *
0000      ;*  COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY   *
0000      ;*  OTHER PERSON.  NO TITLE TO AND OWNERSHIP OF  THE  SOFTWARE IS  HEREBY   *
0000      ;*  TRANSFERRED.                                                            *
0000      ;*                                                                          *
0000      ;*  THE INFORMATION IN THIS SOFTWARE IS  SUBJECT TO CHANGE WITHOUT NOTICE   *
0000      ;*  AND  SHOULD  NOT  BE  CONSTRUED AS  A COMMITMENT BY DIGITAL EQUIPMENT   *
0000      ;*  CORPORATION.                                                            *
0000      ;*                                                                          *
0000      ;*  DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE  OR  RELIABILITY OF ITS   *
0000      ;*  SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.                 *
0000      ;*                                                                          *
0000      ;*                                                                          *
0000      ;****************************************************************************
0000      ;
0000
0000      ;++
0000      ; FACILITY: BASIC code support
0000      ;
0000      ; ABSTRACT:
0000      ;
0000      ;       This module multiplies 2 arrays of any dtype and stores the result in a
0000      ;       third array of any dtype.
0000      ;
0000      ; ENVIRONMENT: User Mode, AST Reentrant
0000      ;
0000      ;--
0000      ; AUTHOR: R. Will, CREATION DATE: 11-Jul-79
0000      ;
0000      ; MODIFIED BY:
0000      ;++
0000      ; 1-001 - Original
0000      ; 1-002 - Change MTH$DFLOOR_R1 to MTH$DFLOOR_R3.  JBS 25-JUL-1979
0000      ; 1-003 - Add check for Illegal Operation error.  RW  28-Sept-79
0000      ; 1-004 - Set IV bit in mask to signal integer overflow.  RW  2-Oct-79
0000      ; 1-005 - Redo scaling.  RW  13-Dec-79
0000      ; 1-006 - Change MTH$DFLOOR_R3 to MTH$DINT_R4.  JBS 19-DEC-1979
0000      ; 1-007 - Fix test for 'same array' for virtual.  RW  15-Feb-1980
0000      ; 1-008 - Add support for byte, g and h floating.  PLL 17-Sep-81
0000      ; 1-009 - More modifications for new data types.  PLL 24-Sep-81
0000      ; 1-010 - Changed shared external reference to G^ RNH 25-Sep-81
0000      ; 1-011 - Substitute a macro for the calls to the array fetch and store
0000      ;         routines.  This should speed things up.  PLL 9-Nov-81
0000      ; 1-012 - Correct a run-time expression in the FETCH and STORE macros.
0000      ;         PLL 20-Jan-82
```

```
0000    ; 1-013 - Do not store an hfloat element in R9.  PLL 15-Feb-82
0000    ; 1-014 - Don't list macro expansions.  PLL 16-Mar-82
0000    ; 1-015 - Fix problem with stack (created by edit 013).  PLL 5-Apr-1982
0000    ; 1-016 - Change order of instructions at STORE_HFLOAT.  PLL 14-Apr-1982
0000    ; 1-017 - Added code to support arrays of descriptors.  LEB 28-JUN-1982.
0000    ; 1-018 - Change own storage to stack storage.  LEB 9-Jul-1982
0000    ; 1-019 - Allow gfloat results to be stored in a double destination, and
0000    ;         vice versa.  PLL 7-Oct-1982
0000    ; 1-020 - fix minor typos in byte*long, word*long, and anything*hfloat.
0000    ;         MDL 15-Oct-1982
0000    ; 1-021 - Use G^ for ALL externals.  SBL 16-Nov-1982
0000    ;--
```

```
0000          .SBTTL  DECLARATIONS
0000    ;
0000    ; INCLUDE FILES:
0000    ;
0000
0000          $DSCDEF                                  ; define descriptor offsets
0000          $SFDEF                                   ; use to get scale
0000
0000    ;
0000    ; EXTERNAL DECLARATIONS:
0000    ;
0000
0000          .DSABL  GBL                              ; Prevent undeclared
0000                                                   ;  symbols from being
0000                                                   ;  automatically global.
0000          .EXTRN  BAS$K_ARGDONMAT                  ; signalled if all 3 blocks
0000                                                   ;  not present in array desc
0000                                                   ;  or dimct = 0
0000          .EXTRN  BAS$K_DATTYPERR                  ; signalled if dtype of array
0000                                                   ; isn't word long float double
0000          .EXTRN  BAS$K_MATDIMERR                  ; signalled if # of dims on any
0000                                                   ;  array isn't 0
0000          .EXTRN  BAS$K_ARRMUSSAM                  ; signalled if cols of src1 not
0000                                                   ;  = rows of src2
0000          .EXTRN  BAS$K_ILLOPE                     ; signalled if dest matrix is
0000                                                   ;  same as either src matrix
0000          .EXTRN  BAS$STO_FA_B_R8                  ; array element store for byte
0000          .EXTRN  BAS$STO_FA_W_R8                  ; array element store for word
0000          .EXTRN  BAS$STO_FA_L_R8                  ; array element store for long
0000          .EXTRN  BAS$STO_FA_F_R8                  ; array element store - float
0000          .EXTRN  BAS$STO_FA_D_R8                  ; array element store - double
0000          .EXTRN  BAS$STO_FA_G_R8                  ; array element store - gfloat
0000          .EXTRN  BAS$STO_FA_H_R8                  ; array element store - hfloat
0000          .EXTRN  BAS$FET_FA_B_R8                  ; array element fetch - byte
0000          .EXTRN  BAS$FET_FA_W_R8                  ; array element fetch - word
0000          .EXTRN  BAS$FET_FA_L_R8                  ; array element fetch - long
0000          .EXTRN  BAS$FET_FA_F_R8                  ; array element fetch - float
0000          .EXTRN  BAS$FET_FA_D_R8                  ; array element fetch - double
0000          .EXTRN  BAS$FET_FA_G_R8                  ; array element fetch - gfloat
0000          .EXTRN  BAS$FET_FA_H_R8                  ; array element fetch - hfloat
0000          .EXTRN  BAS$MAT_REDIM                    ; check if redimensioning of
0000                                                   ;  dest array is necessary, if
0000                                                   ;  so, do it
0000          .EXTRN  BAS$$SCALE_R1                    ; scale for double procision
0000          .EXTRN  MTH$DINT_R4                      ; routine to integerize double
0000          .EXTRN  BAS$$STOP                        ; signal fatal errors
0000          .EXTRN  BAS$FETCH_BFA
0000          .EXTRN  BAS$STORE_BFA
0000
0000    ;
0000    ; MACROS:
0000    ;
0000
0000    ;     $BAS$MAT_MUL    multiply loop algorithm, see next page
0000    ;     FETCH           fetch an element from an array
0000    ;     STORE           store an element into an array
0000
```

```
             0000  ;
             0000  ; EQUATED SYMBOLS:
             0000  ;
             0000
    00000000 0000          upper_bound_k = 0                          ; stack offset for temporary
             0000                                                     ;  for upperbound of inner loop
    00000004 0000          lower_bound_k = 4                          ; stack offset for temporary
             0000                                                     ;  for lowerbound for innerloop
    00000008 0000          upper_bound_j = 8                          ; stack offset for temporary
             0000                                                     ;  for upperbound of middle loop
    0000000C 0000          lower_bound_j = 12                         ; stack offset for temporary
             0000                                                     ;  for lowerbound of middle loop
    00000010 0000          upper_bound_i = 16                         ; stack offset for temporary
             0000                                                     ;  for upperbound of outer loop
    00000014 0000          current_j = 20                             ; stack offset for temporary for
             0000                                                     ;  current value of middle loop
    00000018 0000          current_i = 24                             ; stack offset for temporary for
             0000                                                     ;  current value of outer loop
    0000001C 0000          current_sum = 28                           ; stack offset for temporary for
             0000                                                     ;  summing to get element
    0000002C 0000          scale = 44                                 ; stack offset for temporary for
             0000                                                     ;  scale
    00000034 0000          src1 = 52                                  ; place to store element 1 while
             0000                                                     ;  element 2 is fetched
    00000042 0000          value_desc = 66
    00000042 0000          str_len = 66
    00000044 0000          dtype = 68
    00000045 0000          class = 69
    00000046 0000          pointer = 70
    0000004A 0000          data = 74
    00000018 0000          dsc$l_l1_1 = 24                            ; desc offset if 1 sub
    0000001C 0000          dsc$l_u1_1 = 28                            ; desc offset if 1 sub
    0000001C 0000          dsc$l_l1_2 = 28                            ; desc offset if 2 sub
    00000020 0000          dsc$l_u1_2 = 32                            ; desc offset if 2 sub
    00000024 0000          dsc$l_l2_2 = 36                            ; desc offset if 2 sub
    00000028 0000          dsc$l_u2_2 = 40                            ; desc offset if 2 sub
             0000
             0000  ;
             0000  ; OWN STORAGE:
             0000  ;
             0000
             0000
             0000  ;
             0000  ; PSECT DECLARATIONS:
             0000  ;
         00000000          .PSECT _BAS$CODE PIC, USR, CON, REL, LCL, SHR, -
             0000                          EXE, RD, NOWRT, LONG
             0000
```

```
0000     ;+
0000     ; This macro contains the looping mechanism for accessing all elements of
0000     ; an array.  It also contains all the logic for all the combinations of data
0000     ; types and scaling.  A macro is used to make it easy to maintain the parallel
0000     ; code for all the different data types.
0000     ;-
0000             .MACRO  $BAS$MAT_MUL src1_dtype, src2_dtype ; multiply algorithm
0000
0000     ;+
0000     ; Get scale so if any of the arrays is double, the scale will be there
0000     ;-
0000
0000             MOVL    SF$L_SAVE_FP(FP), R0            ; pass FP to get scale
0000             JSB     G^BAS$$SCALE_R1                 ; get scale in R0 & R1
0000                                                     ; call a BLISS routine because
0000                                                     ; the frame offsets are only
0000                                                     ; defined for BLISS
0000             MOVD    R0, scale(SP)                   ; store the scale
0000
0000     ;+
0000     ; Loop through all the rows of the destination matrix.
0000     ; Row and column upper and lower bounds have been initialized on the stack.
0000     ; Current row (current_i) has been initialized to its lower bound.
0000     ;-
0000
0000     LOOP_I_'src1_dtype'src2_dtype':
0000             MOVL    lower_bound_j(SP), current_j(SP) ; initialize current_j
0000                                                     ;   to lower_bound of j
0000
0000     ;+
0000     ; Loop through all the elements (columns) of the current row of the destination
0000     ; matrix.  Current column (current_j) has been initialized to its lower bound.
0000     ; Column upper bound is on the stack (upper_bound_j).
0000     ;-
0000
0000     LOOP_J_'src1_dtype'src2_dtype':
0000             MOVL    lower_bound_k(SP), R11          ; initialize current_k (R11) to
0000                                                     ;  lower_bound of k
0000             CLRQ    current_sum(SP)                 ; a CLRQ will set the temporary
0000                                                     ;  to 0 for all possible dtypes
0000             CLRQ    current_sum+8(SP)               ; sum could be hfloat (but don't
0000                                                     ;  use h instruction here)
0000     ;+
000C     ; Loop through, summing the products of each element of the ith row of src1
0000     ; and the jth column of  src2.  current_i and current_j are on the stack.
0000     ; Source array element pointer (current_k) has been initialized in R11.
0000     ; Distinguish array by data type so that the correct fetch routine can
0000     ; retrieve the data, the correct multiply and add can be done and the correct
0000     ; store routine can be called.
0000
0000     LOOP_K_'src1_dtype'src2_dtype':
0000
0000     ;+
0000     ; Get the data from the first source array
0000     ;-
0000
0000             MOVL    src1_matrix(AP), R0             ; pointer to 1st src array
```

```
0000            MOVL    current_i(SP), R1               ; current row
0000            MOVL    R11, R2                         ; current col
0000            FETCH   'src1_dtype'                    ; fetch data from src1 array
0000            MOV'src1_dtype' R0, src1(SP)            ; store the 1st array element
0000
0000    ;+
0000    ; Get the data from the second source array
0000    ;-
0000
0000            MOVL    src2_matrix(AP), R0             ; pointer to 2nd src array
0000            MOVL    R11, R1                         ; current row
0000            MOVL    current_j(SP), R2               ; current col
0000            FETCH   'src2_dtype'                    ; fetch data from src2 array
0000
0000    ;+
0000    ; If the data types of the 2 source arrays is the same, do the arithmetic in
0000    ; that data type.  Else convert the data to a common type and multiply and add.
0000    ; If either of the source elements is double, descale and multiply.  Scale and
0000    ; integerize the product before adding it to the sum.
0000    ;-
0000
0000            .IF     IDN     src1_dtype, src2_dtype  ; src arrays are
0000                                                    ; same data type
0000            .IF     IDN     src1_dtype, D           ; both sources are double
0000            MULD2   src1(SP), R0                    ; multiply
0000            DIVD2   scale(SP), R0                   ; remove extra scale from result
0000            CMPD    scale(SP), #1                   ; is the scale 0?
0000            BEQL    1$                              ; yes, do not integerize
0000            JSB     G^MTH$DINT_R4                   ; no, integerize
0000    1$:     ADDD2   R0, current_sum(SP)             ; add to sum
0000            .IFF                                    ; sources same dtype, not double
0000            MUL'src1_dtype'2        src1(SP), R0    ; multiply the source elements
0000            ADD'src1_dtype'2        R0, current_sum(SP) ; add product to current sum
0000            .ENDC                                   ; end of same dtype code
0000            .IFF                                    ; src arrays different dtype
0000            .IF     IDN     src1_dtype, H           ; source 1 is hfloat
0000            CVT'src2_dtype'H        R0, R0          ; cvt array2 to hfloat
0000            MULH2   src1(SP), R0                    ; mult the elements
0000            ADDH2   R0, current_sum(SP)             ; add product to current sum
0000            .IFF
0000            .IF     IDN     src2_dtype, H           ; source 2 is hfloat
0000            CVT'src1_dtype'H        src1(SP), src1(SP) ; cvt array1 to hfloat
0000            MULH2   src1(SP), R0                    ; mult the elements
0000            ADDH2   R0, current_sum(SP)             ; add product to current sum
0000            .IFF
0000            .IF     IDN     src1_dtype, G           ; source 1 is gfloat
0000            .IF     IDN     src2_dtype, D           ; special case if g & dbl
0000            DIVD2   scale(SP), R0                   ; descale src2
0000            CVTDH   R0, R0                          ; cvt src2 to hfloat
0000            CVTGH   src1(SP), src1(SP)              ; cvt src1 to hfloat
0000            MULH2   src1(SP), R0                    ; mult the elements
0000            ADDH2   R0, current_sum(SP)             ; add product to current sum
0000            .IFF                                    ; src2 any type but dbl
0000            CVT'src2_dtype'G        R0, R0          ; cvt src2 to gfloat
0000            MULG2   src1(SP), R0                    ; mult the elements
0000            ADDG2   R0, current_sum(SP)             ; add product to current sum
0000            .ENDC                                   ; end of src1 gfloat
```

```
0000    .IFF
0000    .IF     IDN     src2_dtype, G           ; source 2 is gfloat
0000    .IF     IDN     src1_dtype, D           ; special case gfloat & dbl
0000    DIVD2   scale(SP), src1(SP)             ; descale src1
0000    CVTDH   src1(SP), src1(SP)              ; cvt src1 to hfloat
0000    CVTGH   R0, R0                          ; cvt src2 to hfloat
0000    MULH2   src1(SP), R0                    ; mult the elements
0000    ADDH2   R0, current_sum(SP)             ; add product to current sum
0000    .IFF                                    ; src2 gfloat & src1 not dbl
0000    CVT'src1_dtype'G        src1(SP), src1(SP) ; cvt src1 to gfloat
0000    MULG2   src1(SP), R0                    ; mult the elements
0000    ADDG2   R0, current_sum(SP)             ; add product to current sum
0000    .ENDC                                   ; end of src2 gfloat
0000    .IFF
0000    .IF     IDN     src1_dtype, D           ; source 1 is double
0000                                            ; don't have to worry if src2
0000                                            ;  is gfloat because we already
0000                                            ;  checked
0000    CVT'src2_dtype'D        R0, R0          ; cvt array2 to double & save
0000                                            ;  no scaling needed because in
0000                                            ;  multiply scale is in 2nd src
0000    MULD2   src1(SP), R0                    ; multiply
0000    JSB     G^MTH$DINT_R4                   ; integerize
0000    ADDD2   R0, current_sum(SP)             ; add product to current sum
0000    .IFF                                    ; 1st array not double
0000    .IF     IDN     src2_dtype, D           ; is 2nd src double
0000                                            ; yes, make src1 double & save
0000    CVT'src1_dtype'D        src1(SP), src1(SP) ; make 1st src double
0000                                            ;  no scaling needed because for
0000                                            ;  multiply only 1 number is
0000                                            ;  needed to be scaled.
0000    MULD2   src1(SP), R0                    ; compute the product
0000    JSB     G^MTH$DINT_R4                   ; integerize
0000    ADDD2   R0, current_sum(SP)             ; add product to current sum
0000    .IFF                                    ; no double operands try float
0000    .IF     IDN     src1_dtype, F           ; is 1st element float
0000    CVT'src2_dtype'F        R0, R0          ; make 2nd element float
0000    MULF2   src1(SP), R0                    ; multiply the elements
0000    ADDF2   R0, current_sum(SP)             ; add to current sum
0000    .IFF                                    ; 1st array not float
0000    .IF     IDN     src2_dtype, F           ; is 2nd array float
0000    CVT'src1_dtype'F        src1(SP), src1(SP) ; yes-make 1st element float
0000    MULF2   src1(SP), R0                    ; multiply the elements
0000    ADDF2   R0, current_sum(SP)             ; add to current sum
0000    .IFF                                    ; no double or float, try long
0000    .IF     IDN     src1_dtype, L           ; is 1st array long
0000    CVT'src2_dtype'L        R0, R0          ; make 2nd element long
0000    MULL2   src1(SP), R0                    ; multiply elements
0000    ADDL2   R0, current_sum(SP)             ; add product to current sum
0000    .IFF
0000    .IF     IDN     src2_dtype, L           ; src2 is long
0000    CVT'src1_dtype'L        src1(SP), src1(SP) ; cvt src1 to long
0000    MULL2   src1(SP), R0                    ; multiply the elements
0000    ADDL2   R0, current_sum(SP)             ; add product to current sum
0000    .IFF
0000    .IF     IDN     src1_dtype, W           ; src1 is word
0000    CVT'src2_dtype'W        R0, R0          ; cvt src2
```

```
0000            MULW2   src1(SP), R0                    ; mult the elements
0000            ADDW2   R0, current_sum(SP)             ; add product to current sum
0000            .IFF
0000            .IF     IDN     src1_dtype, B           ; src1 is byte
0000            CVT'src2_dtype'B        R0, R0          ; cvt src2
0000            MULB2   src1(SP), R0                    ; mult the elements
0000            ADDB2   R0, current_sum(SP)             ; add product to current sum
0000            .IFF                                    ; last case - src2 must be byte
0000            CVT'src1_dtype'B        src1(SP), src1(SP) ; cvt src1
0000            MULB2   src1(SP), R0                    ; mult the elements
0000            ADDB2   R0, current_sum(SP)             ; add product to current sum
0000            .ENDC
0000            .ENDC
0000            .ENDC
0000            .ENDC
0000            .ENDC
0000            .ENDC
0000            .ENDC
0000            .ENDC
0000            .ENDC
0000            .ENDC
0000            .ENDC
0000            .ENDC
0000            .ENDC
0000
0000    ;+
0000    ; Have multiplied next set of elements and added it to current sum.  See if
0000    ; it is the last product of the sum.  If not continue with current sum.
0000    ; Otherwise, store the sum in the destination array by calling a subroutine
0000    ; (pass pointer to dest in R10 and pointer to stack in R5)
0000    ; and continue with next destination element.
0000    ;-
0000
0000            INCL    R11                             ; get next K
0000            CMPL    R11, upper_bound_k(SP)          ; see if last product in sum
0000            BGTR    5$
0000            BRW     LOOP_K_'src1_dtype'src2_dtype   ; no, continue inner loop
0000
0000                                                    ; finished inner loop so store
0000    5$:     MOVL    SP, R5                          ; point to temps
0000            .IF     IDN     src1_dtype, src2_dtype  ; src arrays are
0000                                                    ; same data type
0000            BSBW    DEST_CASE_'src1_dtype'          ; go cvrt to dest type and store
0000            .IFF                                    ; src arrays different dtype
0000            .IF     IDN     src1_dtype, H           ; source 1 is hfloat
0000            BSBW    DEST_CASE_H                     ; cvt from hfloat to dest type
0000            .IFF
0000            .IF     IDN     src2_dtype, H           ; source 2 is hfloat
0000            BSBW    DEST_CASE_H                     ; cvt from hfloat to dest type
0000            .IFF
0000            .IF     IDN     src1_dtype, G           ; source 1 is gfloat
0000            .IF     IDN     src2_dtype, D           ; special case gfloat & dbl
0000            BSBW    DEST_CASE_H                     ; ops done in hfloat so cvt
0000                                                    ;  from hfloat to dest type
0000            .IFF                                    ; gfloat & all other dtypes
0000            BSBW    DEST_CASE_G                     ; cvt from gfloat to dest type
0000            .ENDC
```

```
0000  405         .IFF
0000  406         .IF     IDN     src2_dtype, G          ; source 2 is gfloat
0000  407         .IF     IDN     src1_dtype, D          ; special case dbl & gfloat
0000  408         BSBW    DEST_CASE_H                    ; ops done in hfloat so cvt
0000  409                                                ;  from hfloat to dest type
0000  410         .IFF                                   ; gfloat & all other dtypes
0000  411         BSBW    DEST_CASE_G                    ; cvt from gfloat to dest type
0000  412         .ENDC
0000  413         .IFF
0000  414         .IF     IDN     src1_dtype, D          ; source 1 is double
0000  415         BSBW    DEST_CASE_D                    ; cnvrt from double to dest type
0000  416                                                ; and store
0000  417                                                ; (note that we don't have to
0000  418                                                ;  worry about dbl & gfloat here
0000  419                                                ;  because it was handled above)
0000  420         .IFF                                   ; 1st array not double
0000  421         .IF     IDN     src2_dtype, D          ; is 2nd src double
0000  422                                                ; yes, make src1 double & save
0000  423         BSBW    DEST_CASE_D                    ; cnvrt from double to dest type
0000  424                                                ; and store
0000  425         .IFF                                   ; no double operands try float
0000  426         .IF     IDN     src1_dtype, F          ; is 1st element float
0000  427         BSBW    DEST_CASE_F                    ; cnvrt from float to dest type
0000  428                                                ; and store
0000  429         .IFF                                   ; 1st array not float
0000  430         .IF     IDN     src2_dtype, F          ; is 2nd array float
0000  431         BSBW    DEST_CASE_F                    ; cnvrt from float to dest type
0000  432                                                ; and store
0000  433         .IFF
0000  434         .IF     IDN     src'_dtyoe, L          ; source 1 is long
0000  435         BSBW    DEST_CASE_L                    ; cnvrt from long to dest type
0000  436                                                ; and store
0000  437         .IFF
0000  438         .IF     IDN     src2_dtype, L          ; source 2 is long
0000  439         BSBW    DEST_CASE_L                    ; cvt from long to dest type
0000  440         .IFF
0000  441         .IF     IDN     src1_dtype, W          ; source 1 is word
0000  442         BSBW    DEST_CASE_W                    ; cvt from word to dest type
0000  443         .IFF
0000  444         .IF     IDN     src2_dtype, W          ; source 2 is word
0000  445         BSBW    DEST_CASE_W                    ; cvt from word to dest type
0000  446                                                ; byte and any other data type
0000  447                                                ; would've been caught by one
0000  448                                                ; of the above cases, and byte
0000  449                                                ; & byte is handles by case 1
0000  450         .IFF
0000  451         .IF     IDN     src1_dtype, B          ; source 1 is byte
0000  452         BSBW    DEST_CASE_B
0000  453         .IFF
0C00  454         BSBW    DEST_CASE_B                    ; only thing left
0000  455         .ENDC
0000  456         .ENDC
0000  457         .ENDC
0000  458         .ENDC
0000  459         .ENDC
0000  460         .ENDC
0000  461         .ENDC
```

```
0000            .ENDC
0000            .ENDC
0000            .ENDC
0000            .ENDC
0000            .ENDC
0000            .ENDC
0000            .ENDC
0000
0000    ;+
0000    ; Have stored that element.  Now see if it was the last column.  If not,
0000    ; continue with the next column.  Otherwise continue to next row.
0000    ;-
0000
0000            INCL    current_j(SP)                       ; get next column
0000            CMPL    current_j(sp), upper_bound_j(SP)    ; see if last column done
0000            BGTR    20$
0000            BRW     LOOP_J_'src1_dtype'src2_dtype'      ; no, continue inner loop
0000
0000    ;+
0000    ; Have completed entire row.  See if it was the last row.  If not,
0000    ; continue with next row.
0000    ;-
0000
0000    20$:    INCL    current_i(SP)                       ; get next row
0000            CMPL    current_i(SP), upper_bound_i(SP)    ; see if last row done
0000            BGTR    10$
0000            BRW     LOOP_I_'src1_dtype'src2_dtype'      ; no, continue outer loop
0000
0000    10$:    RET                                         ; yes, finished
0000
0000            .ENDM
```

```
                  0000          .SBTTL  BAS$MAT_MUL  - Multiply 2 arrays giving a third
                  0000  ;++
                  0000  ; FUNCTIONAL DESCRIPTION:
                  0000  ;
                  0000  ;       Multiply 2 arrays giving a third.  Signal an error if the upper and
                  0000  ;       lower bounds (excluding 0) for columns in src1_matrix does not equal
                  0000  ;       the upper and lower bounds (excluding 0) for rows in src2_matrix.
                  0000  ;       An error will also be signalled if any of the three matrices does not
                  0000  ;       have a DIMCT of 2, or if DSC$A_POINTER in either src1_matrix or
                  0000  ;       src2_matrix is the same as DSC$A_POINTER of dest_matrix.
                  0000  ;       Redimension the output to have a lower bound of 0 for both dimensions,
                  0000  ;       and an upper bound for rows equal to the upper bound for rows for
                  0000  ;       src1_matrix, and an upper bound for columns equal to the upper bound
                  0000  ;       for columns for src2_matrix.  Initialize all the necessary
                  0000  ;       looping information on the stack.  Conversions may have to be done
                  0000  ;       so that the sources are the same data type, so divide
                  0000  ;       the looping portion according to the data types.  Conversion to the
                  0000  ;       correct destination data type will be done by a JSB to a routine,
                  0000  ;       instead of multiplying the number of possible combinations by 4.
                  0000  ;
                  0000  ; CALLING SEQUENCE:
                  0000  ;
                  0000  ;       CALL BAS$MAT_MUL (src1_array.rx.da, src2_array.rw.da, dest_matrix.wx.da)
                  0000  ;
                  0000  ; INPUT PARAMETERS:
                  0000  ;
        00000004  0000          src1_matrix = 4
        00000008  0000          src2_matrix = 8
                  0000  ;
                  0000  ; IMPLICIT INPUTS:
                  0000  ;
                  0000  ;       Scale from the callers frame to scale double precision.
                  0000  ;
                  0000  ; OUTPUT PARAMETERS:
                  0000  ;
        0000000C  0000          dest_matrix = 12
                  0000  ;
                  0000  ; IMPLICIT OUTPUTS:
                  0000  ;
                  0000  ;       NONE
                  0000  ;
                  0000  ; FUNCTION VALUE:
                  0000  ; COMPLETION CODES:
                  0000  ;
                  0000  ;       NONE
                  0000  ;
                  0000  ; SIDE EFFECTS:
                  0000  ;
                  0000  ;       This routine calls the redimensioning routine and the array element
                  0000  ;       fetch and store routines and therefore may signal any of their errors.
                  0000  ;       It may also signal any of the errors listed in the externals section.
                  0000  ;       It may also cause the destination array to have different dimensions.
                  0000  ;
                  0000  ;--
                  0000
            4FFC  0000          .ENTRY BAS$MAT_MUL, ^M<R2,R3,R4,R5,R6,R7,R8,R9,R10,R11,IV>
                  0002
```

```
                              0002   551 ;+
                              0002   552 ;        REGISTER USAGE
                              0002   553 ;        R0 - R8 destroyed by store routines
                              0002   554 ;        R9      not used
                              0002   555 ;        R10     pointer to dest matrix descriptor (except for double in which
                              0002   556 ;                case R10 is part of double value R9-R10)
                              0002   557 ;        R11     current value of inner subscript
                              0002   558 ;-
                              0002   559
                              0002   560 ;+
                              0002   561 ; Put routine arguments into registers for ease of use.
                              0002   562 ; If block 2 of array descriptor (multipliers) is not present then error.
                              0002   563 ;-
                              0002   564
       52    04 AC      D0    0002   565          MOVL    src1_matrix(AP), R2              ; ptr to src1 array descr
   36 0A A2     07      E1    0006   566          BBC     #DSC$V_FL_BOUNDS, DSC$B_AFLAGS(R2), ERR_ARGDONMAT
                              000B   567                                                   ; exit if block 3 not
                              000B   568                                                   ; present in descriptor
       02    0B A2      91    000B   569          CMPB    DSC$B_DIMCT(R2), #2              ; 2 dimensional?
                23      12    000F   570          BNEQU   ERR_MATDIMERR                    ; if not, error
       53    08 AC      D0    0011   571          MOVL    src2_matrix(AP), R3              ; ptr to src2 array descr
   27 0A A3     07      E1    0015   572          BBC     #DSC$V_FL_BOUNDS, DSC$B_AFLAGS(R3), ERR_ARGDONMAT
                              001A   573                                                   ; exit if block 3 not
                              001A   574                                                   ; present in descriptor
       5A    0C AC      D0    001A   575          MOVL    dest_matrix(AP), R10             ; pointer to dest descriptor
                7E      7C    001E   576          CLRQ    -(SP)                            ; save space for VALUE_DESC
                7E      7C    0020   577          CLRQ    -(SP)                            ;          AND
                7E      7C    0022   578          CLRQ    -(SP)                            ; DATA
                7E      7C    0024   579          CLRD    -(SP)                            ; space for src1 element
                7E      7C    0026   580          CLRD    -(SP)                            ;  it may be hfloat
                7E      73    0028   581          TSTD    -(SP)                            ; save space for scale
                7E      7C    002A   582          CLRD    -(SP)                            ; reserve space to save sum
                7E      7C    002C   583          CLRD    -(SP)                            ;  possible hfloat sum
       02    0B A3      91    002E   584          CMPB    DSC$B_DIMCT(R3), #2              ; 2 dimensional?
                34      13    0032   585          BEQLU   INIT_TWO_SUBS                    ; if 2-D continue, else
                              0034   586                                                   ; fall into error
                              0034   587
                              0034   588 ERR_MATDIMERR:
         00000000'8F    DD    0034   589          PUSHL   #BAS$K_MATDIMERR                 ; Signal error, src arrays
      00000000'GF   01  FB    003A   590          CALLS   #1, G^BAS$$STOP                  ;  don't have same # dimensns
                              0041   591
                              0041   592 ERR_ARGDONMAT:
         00000000'8F    DD    0041   593          PUSHL   #BAS$K_ARGDONMAT                 ; signal error, 0 for dimct
      00000000'GF   01  FB    0047   594          CALLS   #1, G^BAS$$STOP                  ; or block 2 or 3 absent
                              004E   595
                              004E   596 ERR_ARRMUSSAM:
         00000000'8F    DD    004E   597          PUSHL   #BAS$K_ARRMUSSAM                 ; Signal error, src arrays
      00000000'GF   01  FB    0054   598          CALLS   #1, G^BAS$$STOP                  ;  same bounds
                              005B   599
                              005B   600 ERR_ILLOPE:
         00000000'8F    DD    005B   601          PUSHL   #BAS$K_ILLOPE                    ; Signal error, dest array is
      00000000'GF   01  FB    0061   602          CALLS   #1, G^BAS$$STOP                  ;  as one of source arrays
                              0068   603
                              0068   604 ;+
                              0068   605 ; There are 2 subscripts.  Check and redimension the destination array if
                              0068   606 ; necessary.  Put the upper bound for both subscripts on the
                              0068   607 ; stack and make sure that the lower bound for both subscripts will start
```

```
                          0068   608 ; at 1 (do not alter row or col 0)
                          0068   609 ;-
                          0068   610
                          0068   611 INIT_TWO_SUBS:
    20 A3    28 A2    91  0068   612         CMPB    dsc$l_u2_2(R2), dsc$l_u1_2(R3)  ; does src1 array have the same
                          006D   613                                                 ;  2nd upper bound as src2 array
                          006D   614                                                 ;  has for 1st upper bound
             DF       12  006D   615         BNEQU   ERR_ARRMUSSAM                   ; no, error
    1C A3    24 A2    91  006F   616         CMPB    dsc$l_l2_2(R2), dsc$l_l1_2(R3)  ; does src1 array have the same
                          0074   617                                                 ;  2nd lower bound as src2 array
                          0074   618                                                 ;  has for 1st lower bound
             D8       12  0074   619         BNEQU   ERR_ARRMUSSAM                   ; no, error
          28 A3       DD  0076   620         PUSHL   dsc$l_u2_2(R3)                  ; 2nd upper bound
          20 A2       DD  0079   621         PUSHL   dsc$l_u1_2(R2)                  ; 1st upper bound
             5A       DD  007C   622         PUSHL   R10                             ; dest array pointer
 00000000'GF 03       FB  007E   623         CALLS   #3, G^BAS$MAT_REDIM             ; redimension destination
       04    03 AA    91  0085   624         CMPB    DSC$B_CLASS(R10), #DSC$K_CLASS_A        ; is array virtual?
             10       12  0089   625         BNEQU   VIRTUAL_SAME                    ; yes, go check virtual
                          008B   626                                                 ; if dest not virtual even if
                          008B   627                                                 ;  src is virtual pointer will
                          008B   628                                                 ;  be 0 and won't match
    04 AA    04 A2    D1  008B   629         CMPL    DSC$A_POINTER(R2), DSC$A_POINTER(R10) ; is dest same as src1
             C9       13  0090   630         BEQLU   ERR_ILLOPE                      ; yes, error
    04 AA    04 A3    D1  0092   631         CMPL    DSC$A_POINTER(R3), DSC$A_POINTER(R10) ; is dest same as src2
             C2       13  0097   632         BEQLU   ERR_ILLOPE                      ; yes, error
             28       11  0099   633         BRB     INIT_STACK
                          009B   634 VIRTUAL_SAME:
       04    03 A2    91  009B   635         CMPB    DSC$B_CLASS(R2), #DSC$K_CLASS_A ; is src1 virtual?
             0E       13  009F   636         BEQLU   1$                              ; no, cant be same
    FC AA    FC A2    D1  00A1   637         CMPL    DSC$L_LOGUNIT(R2), DSC$L_LOGUNIT(R10) ; is dest same as src1?
             07       12  00A6   638         BNEQ    1$                              ; no
    F8 AA    F8 A2    D1  00A8   639         CMPL    DSC$L_BYTEOFF(R2), DSC$L_BYTEOFF(R10) ; is dest same as src1?
                          00AD   640                                                 ; (check logunit and byteoff)
             AC       13  00AD   641         BEQL    ERR_ILLOPE                      ; yes error
       04    03 A3    91  00AF   642 1$:     CMPB    DSC$B_CLASS(R3), #DSC$K_CLASS_A ; is src2 virtual?
             0E       13  00B3   643         BEQLU   INIT_STACK                      ; no, can't be same
    FC AA    FC A2    D1  00B5   644         CMPL    DSC$L_LOGUNIT(R2), DSC$L_LOGUNIT(R10) ; is dest same as src1?
             07       12  00BA   645         BNEQ    INIT_STACK
    F8 AA    F8 A3    D1  00BC   646         CMPL    DSC$L_BYTEOFF(R3), DSC$L_BYTEOFF(R10) ; is dest same as src2?
             98       13  00C1   647         BEQL    ERR_ILLOPE                      ; yes error
                          00C3   648 INIT_STACK:
          1C A2       DD  00C3   649         PUSHL   dsc$l_l1_2(R2)                  ; initialize current_i counter
             03       14  00C6   650         BGTR    1$                              ; not row 0 or neg
       6E    01       D0  00C8   651         MOVL    #1, (SP)                        ; start with 1
             7E       D4  00CB   652 1$:     CLRL    -(SP)                           ; save space for current j
          20 A2       DD  00CD   653         PUSHL   dsc$l_u1_2(R2)                  ; upper_bound_i
          24 A3       DD  00D0   654         PUSHL   dsc$l_l2_2(R3)                  ; lower_bound_j
             03       14  00D3   655         BGTR    2$                              ; not row 0 or neg, do cols
       6E    01       D0  00D5   656         MOVL    #1, (SP)                        ; start with row 1
          28 A3       DD  00D8   657 2$:     PUSHL   dsc$l_u2_2(R3)                  ; upper_bound_j
          24 A2       DD  00DB   658         PUSHL   dsc$l_l2_2(R2)                  ; lower_bound_k
             03       14  00DE   659         BGTR    3$                              ; not col 0 or neg
       6E    01       D0  00E0   660         MOVL    #1, (SP)                        ; start with k=1
          28 A2       DD  00E3   661 3$:     PUSHL   dsc$l_u2_2(R2)                  ; upper_bound_k
                          00E6   662
                          00E6   663 ;+
                          00E6   664 ; Algorithm now differs according to data types
```

```
                             00E6  ;-
                             00E6
                             00E6  SEPARATE_DTYPES:
                             00E6
     05    06    02 A2    8F  00E6  5$:     CASEB   DSC$B_DTYPE(R2), #DSC$K_DTYPE_B, #<DSC$K_DTYPE_D - DSC$K_DTYPE_B>
                       0037'  00EB  2$:     .WORD   BYTE-2$                                ; code for byte dtype
                       0F61'  00ED          .WORD   WORD-2$                                ; code for word dtype
                       1E8B'  00EF          .WORD   LONG-2$                                ; code for long dtype
                       002A'  00F1          .WORD   ERR_DATTYPERR-2$                       ; quad not supported
                       2DB3'  00F3          .WORD   FLOAT-2$                               ; code for flcat dtype
                       3CD9'  00F5          .WORD   DOUBLE-2$                              ; code for double dtype
                             00F7
                             00F7  ;+
                             00F7  ; G and H floating fall outside the range of the CASEB.
                             00F7  ;-
                             00F7
           1B    02 A2    91  00F7          CMPB    DSC$B_DTYPE(R2), #DSC$K_DTYPE_G
                    03    12  00FB          BNEQ    3$
                  4C13    31  00FD          BRW     GFLOAT                                 ; code for gfloat dtype
                             0100
           1C    02 A2    91  0100  3$:     CMPB    DSC$B_DTYPE(R2), #DSC$K_DTYPE_H
                    03    12  0104          BNEQ    4$
                  5B6F    31  0106          BRW     HFLOAT                                 ; code for hfloat dtype
                             0109
           18    02 A2    91  0109  4$:     CMPB    DSC$B_DTYPE(R2), #DSC$K_DTYPE_DSC
                    06    12  010D          BNEQ    ERR_DATTYPERR
           52    04 A2    D0  010F          MOVL    4(R2), R2                              ; R2 <-- addr of descriptor
                    D1    11  0113          BRB     5$                                     ; CASE again on dtype in desc
                             0115
                             0115  ERR_DATTYPERR:
       00000000'8F        DD  0115          PUSHL   #BAS$K_DATTYPERR                       ; Signal error, unsupported
   00000000'GF      01    FB  0118          CALLS   #1, G^BAS$$STOP                        ; dtype in array desc
```

```
                          0122   699 ;+
                          0122   700 ; Source1 array is a byte array.  Now differentiate on the source2 type.
                          0122   701 ;-
                          0122   702
 05   06   02 A3     8F   0122   703 BYTE:    CASEB   DSC$B_DTYPE(R3), #DSC$K_DTYPE_B, #<DSC$K_DTYPE_D - DSC$K_DTYPE_B>
                  002D'   0127   704 1$:      .WORD   BYTE_TO_BYTE-1$                    ; code for byte dtype
                  0249'   0129   705          .WORD   BYTE_TO_WORD-1$                    ; code for word dtype
                  0468'   012B   706          .WORD   BYTE_TO_LONG-1$                    ; code for long dtype
                  FFEE    012D   707          .WORD   ERR_DATTYPERR-1$                   ; quad not supported
                  0689'   012F   708          .WORD   BYTE_TO_FLOAT-1$                   ; code for float dtype
                  0BAA'   0131   709          .WORD   BYTE_TO_DOUBLE-1$                  ; code for double dtype
                          0133   710
                          0133   711 ;+
                          0133   712 ; G and H floating fall outside the range of the CASEB.
                          0133   713 ;-
                          0133   714
      1B   02 A3     91   0133   715          CMPB    DSC$B_DTYPE(R3), #DSC$K_DTYPE_G
              03     12   0137   716          BNEQ    2$
            0ABC     31   0139   717          BRW     BYTE_TO_GFLOAT
                          013C   718
      1C   02 A3     91   013C   719 2$:      CMPB    DSC$B_DTYPE(R3), #DSC$K_DTYPE_H
              03     12   0140   720          BNEQ    3$
            0CDD     31   0142   721          BRW     BYTE_TO_HFLOAT
                          0145   722
      18   02 A3     91   0145   723 3$:      CMPB    DSC$B_DTYPE(R3), #DSC$K_DTYPE_DSC
              06     12   0149   724          BNEQ    4$
      53   04 A3     D0   014B   725          MOVL    4(R3), R3                          ; R3 <-- addr of descriptor
              D1     11   014F   726          BRB     BYTE                               ; CASE again on dtype in desc
                          0151   727
            FFC1     31   0151   728 4$:      BRW     ERR_DATTYPERR
                          0154   729 ;+
                          0154   730 ; Now type of source1 and source2 arrays are known.  Use the macro to
                          0154   731 ; generate the code for each case
                          0154   732 ;-
                          0154   733
```

```
     0154    735 BYTE_TO_BYTE:   $BAS$MAT_MUL    B, B
     0370    736
```

```
0370     BYTE_TO_WORD:   $BAS$MAT_MUL    B, W
058F
```

```
     058F    741 BYTE_TO_LONG:   $BAS$MAT_MUL    B, L
     07B0    742
```

```
        07B0   744 BYTE_TO_FLOAT:  $BAS$MAT_MUL    B, F
        09D1   745
```

```
09D1   747 BYTE_TO_DOUBLE: $BAS$MAT_MUL    B, D
0BF8   748
```

```
0BF8   750 BYTE_TO_GFLOAT: $BAS$MAT_MUL    B, G
0E22   751
```

```
     0E22    753 BYTE_TO_HFLOAT: $BAS$MAT_MUL    B, H
```

```
                         104C  ;+
                         104C  ; Source1 array is a word array.  Now differentiate on the source2 type.
                         104C  ;-
                         104C
 05    06    02 A3   8F  104C  WORD:   CASEB   DSC$B_DTYPE(R3), #DSC$K_DTYPE_B, #<DSC$K_DTYPE_D - DSC$K_DTYPE_B>
                 002D'   1051  1$:     .WORD   WORD_TO_BYTE-1$                 ; code for byte dtype
                 024C'   1053          .WORD   WORD_TO_WORD-1$                 ; code for word dtype
                 0468'   1055          .WORD   WORD_TO_LONG-1$                 ; code for long dtype
                 F0C4    1057          .WORD   ERR_DATTYPERR-1$                ; quad not supported
                 0689'   1059          .WORD   WORD_TO_FLOAT-1$                ; code for float dtype
                 08AA'   105B          .WORD   WORD_TO_DOUBLE-1$               ; code for double dtype
                         105D
                         105D  ;+
                         105D  ; G and H floating fall outside the range of the CASEB.
                         105D  ;-
                         105D
       1B    02 A3   91  105D          CMPB    DSC$B_DTYPE(R3), #DSC$K_DTYPE_G
                03   12  1061          BNEQ    2$
              0ABC   31  1063          BRW     WORD_TO_GFLOAT                  ; code for gfloat dtype
                         1066
       1C    02 A3   91  1066  2$:     CMPB    DSC$B_DTYPE(R3), #DSC$K_DTYPE_H
                03   12  106A          BNEQ    3$
              0CDD   31  106C          BRW     WORD_TO_HFLOAT                  ; code for hfloat dtype
                         106F
       18    02 A3   91  106F  3$:     CMPB    DSC$B_DTYPE(R3), #DSC$K_DTYPE_DSC
                06   12  1073          BNEQ    4$
       53    04 A3   D0  1075          MOVL    4(R3), R3                       ; R3 <-- addr of descriptor
                D1   11  1079          BRB     WORD                            ; CASE again on dtype in desc
                         107B
              F097   31  107B  4$:     BRW     ERR_DATTYPERR
                         107E
                         107E  ;+
                         107E  ; Now type of source1 and source2 arrays are known.  Use the macro to
                         107E  ; generate the code for each case
                         107E  ;-
                         107E
```

```
     107E   792 WORD_TO_BYTE:   $BAS$MAT_MUL    W, B
     129D   793
```

```
        129D    795 WORD_TO_WORD:   $BAS$MAT_MUL    W, W
        14B9    796
```

```
     14B9    798 WORD_TO_LONG:   $BAS$MAT_MUL    W, L
     16DA    799
```

```
16DA   801 WORD_TO_FLOAT:  $BAS$MAT_MUL    W, F
18FB   802
```

```
     18FB   804 WORD_TO_DOUBLE: $BAS$MAT_MUL    W, D
     1B22   805
     1B22    807 WORD_TO_GFLOAT: $BAS$MAT_MUL    W, G
     1D4C    808
     1D4C    810 WORD_TO_HFLOAT: $BAS$MAT_MUL     W, H
```

```
                        1F76      ;+
                        1F76      ; Source1 array is a longword array.  Now differentiate on the source2 type
                        1F76      ;-
                        1F76
05    06    02 A3    8F 1F76      LONG:     CASEB     DSC$B_DTYPE(R3), #DSC$K_DTYPE_B, #<DSC$K_DTYPE_D - DSC$K_DTYPE_B>
                  002D' 1F7B      1$:       .WORD     LONG_TO_BYTE-1$                      ; code for byte dtype
                  024C' 1F7D                .WORD     LONG_TO_WORD-1$                      ; code for word dtype
                  046B' 1F7F                .WORD     LONG_TO_LONG-1$                      ; code for long dtype
                  E19A  1F81                .WORD     ERR_DATTYPERR-1$                     ; quad not supported
                  0687' 1F83                .WORD     LONG_TO_FLOAT-1$                     ; code for float dtype
                  08A8' 1F85                .WORD     LONG_TO_DOUBLE-1$                    ; code for double dtype
                        1F87
                        1F87
                        1F87      ;+
                        1F87      ; G and H floating fall outside the range of the CASEB.
                        1F87      ;-
                        1F87
      1B    02 A3    91 1F87                CMPB      DSC$B_DTYPE(R3), #DSC$K_DTYPE_G
               03    12 1F8B                BNEQ      2$
             0ABA    31 1F8D                BRW       LONG_TO_GFLOAT                       ; code for gfloat dtype
                        1F90
      1C    02 A3    91 1F90      2$:       CMPB      DSC$B_DTYPE(R3), #DSC$K_DTYPE_H
               03    12 1F94                BNEQ      3$
             0CDB    31 1F96                BRW       LONG_TO_HFLOAT                       ; code for hfloat dtype
                        1F99
      18    02 A3    91 1F99      3$:       CMPB      DSC$B_DTYPE(R3), #DSC$K_DTYPE_DSC
               06    12 1F9D                BNEQ      4$
      53    04 A3    D0 1F9F                MOVL      4(R3), R3                            ; R3 <-- addr of descriptor
               D1    11 1FA3                BRB       LONG                                 ; CASE again on dtype in desc
                        1FA5
             E16D    31 1FA5      4$:       BRW       ERR_DATTYPERR
                        1FA8      ;+
                        1FA8      ; Now type of source1 and source2 arrays are known.  Use the macro to
                        1FA8      ; generate the code for each case
                        1FA8      ;-
```

```
     1FA8    848 LONG_TO_BYTE:   $BAS$MAT_MUL    L, B
     21C7    849
```

```
          21C7    051 LONG_TO_WORD:   $BAS$MAT_MUL    L, W
          23E6    052
```

```
    23E6   854 LONG_TO_LONG:   $BAS$MAT_MUL    L, L
    2602   855
```

```
     2602   857 LONG_TO_FLOAT:  $BAS$MAT_MUL    L, F
     2823   858
     2823    860 LONG_TO_DOUBLE: $BAS$MAT_MUL    L, D
     2A4A    861
     2A4A    863 LONG_TO_GFLOAT: $BAS$MAT_MUL     L, G
     2C74    864
     2C74    866 LONG_TO_HFLOAT: $BAS$MAT_MUL    L, H
     2E9E    867
```

```
                        2E9E   869 ;+
                        2E9E   870 ; Source1 array is a floating array.  Now differentiate on the source2 type
                        2E9E   871 ;-
                        2E9E   872
  05   06   02 A3   8F  2E9E   873 FLOAT:   CASEB   DSC$B_DTYPE(R3), #DSC$K_DTYPE_B, #<DSC$K_DTYPE_D - DSC$K_DTYPE_B>
               002D'    2EA3   874 1$:      .WORD   FLOAT_TO_BYTE-1$                      ; code for byte dtype
               024C'    2EA5   875          .WORD   FLOAT_TO_WORD-1$                      ; code for word dtype
               046B'    2EA7   876          .WORD   FLOAT_TO_LONG-1$                      ; code for long dtype
               D272     2EA9   877          .WORD   ERR_DATTYPERR-1$                      ; quad not supported
               068A'    2EAB   878          .WORD   FLOAT_TO_FLOAT-1$                     ; code for float dtype
               08A6'    2EAD   879          .WORD   FLOAT_TO_DOUBL-1$                     ; code for double dtype
                        2EAF   880
                        2EAF   881 ;+
                        2EAF   882 ; G and H floating fall outside the range of the CASEB.
                        2EAF   883 ;-
                        2EAF   884
       1B   02 A3   91  2EAF   885          CMPB    DSC$B_DTYPE(R3), #DSC$K_DTYPE_G
               03   12  2EB3   886          BNEQ    2$
             0AB8   31  2EB5   887          BRW     FLOAT_TO_GFLOA
                        2EB8   888
       1C   02 A3   91  2EB8   889 2$:      CMPB    DSC$B_DTYPE(R3), #DSC$K_DTYPE_H
               03   12  2EBC   890          BNEQ    3$
             0CD9   31  2EBE   891          BRW     FLOAT_TO_HFLOA
                        2EC1   892
       18   02 A3   91  2EC1   893 3$:      CMPB    DSC$B_DTYPE(R3), #DSC$K_DTYPE_DSC
               06   12  2EC5   894          BNEQ    4$
       53   04 A3   D0  2EC7   895          MOVL    4(R3), R3                             ; R3 <-- addr of descriptor
               D1   11  2ECB   896          BRB     FLOAT                                 ; CASE again on dtype in desc
                        2ECD   897
             D245   31  2ECD   898 4$:      BRW     ERR_DATTYPERR
                        2ED0   899 ;+
                        2ED0   900 ; Now type of source1 and source2 arrays are known.  Use the macro to
                        2ED0   901 ; generate the code for each case
                        2ED0   902 ;-
```

```
2ED0     FLOAT_TO_BYTE:  $BAS$MAT_MUL    F, B
30EF
```

```
         30EF    907 FLOAT_TO_WORD:  $BAS$MAT_MUL    F, W
         330E    908
```

```
330E   910 FLOAT_TO_LONG:  $BAS$MAT_MUL    F, L
352D   911
```

```
      352D    913 FLOAT_TO_FLOAT: $BAS$MAT_MUL    F, F
      3749    914
```

```
     3749    916 FLOAT_TO_DOUBL: $BAS$MAT_MUL    F, D
     3970    917
```

```
        3970    919 FLOAT_TO_GFLOA: $BAS$MAT_MUL    F, G
        3B9A    920
        3B9A    922 FLOAT_TO_HFLOA: $BAS$MAT_MUL    F, H
        3DC4    923
```

```
                          3DC4    925 ;+
                          3DC4    926 ; Source1 array is a double array.  Now differentiate on the source2 type.
                          3DC4    927 ;-
                          3DC4    928
       05   06   02 A3   8F  3DC4    929 DOUBLE: CASEB    DSC$B_DTYPE(R3), #DSC$K_DTYPE_B, #<DSC$K_DTYPE_D - DSC$K_DTYPE_B>
                     002D'  3DC9    930 1$:      .WORD    DOUBLE_TO_BYTE-1$                ; code for byte dtype
                     0252'  3DCB    931          .WORD    DOUBLE_TO_WORD-1$                ; code for word dtype
                     0477'  3DCD    932          .WORD    DOUBLE_TO_LONG-1$                ; code for long dtype
                     C34C   3DCF    933          .WORD    ERR_DATTYPERR-1$                 ; quad not supported
                     069C'  3DD1    934          .WORD    DOUBLE_TO_FLOA-1$                ; code for float dtype
                     08C1'  3DD3    935          .WORD    DOUBLE_TO_DOUBL-1$               ; code for double dtype
                          3DD5    936
                          3DD5    937 ;+
                          3DD5    938 ; G and H floating fall outside the range of the CASEB.
                          3DD5    939 ;-
                          3DD5    940
            1B   02 A3   91  3DD5    941          CMPB     DSC$B_DTYPE(R3), #DSC$K_DTYPE_G
                    03   12  3DD9    942          BNEQ     2$
                  0AD8   31  3DDB    943          BRW      DOUBLE_TO_GFLOA
                          3DDE    944
            1C   02 A3   91  3DDE    945 2$:      CMPB     DSC$B_DTYPE(R3), #DSC$K_DTYPE_H
                    03   12  3DE2    946          BNEQ     3$
                  0D02   31  3DE4    947          BRW      DOUBLE_TO_HFLOA
                          3DE7    948
            18   02 A3   91  3DE7    949 3$:      CMPB     DSC$B_DTYPE(R3), #DSC$K_DTYPE_DSC
                    06   12  3DEB    950          BNEQ     4$
            53   04 A3   D0  3DED    951          MOVL     4(R3), R3                        ; R3 <-- addr of descriptor
                    D1   11  3DF1    952          BRB      DOUBLE                           ; CASE again on dtype in desc
                          3DF3    953
                  C31F   31  3DF3    954 4$:      BRW      ERR_DATTYPERR
                          3DF6    955 ;+
                          3DF6    956 ; Now type of source1 and source2 arrays are known.  Use the macro to
                          3DF6    957 ; generate the code for each case
                          3DF6    958 ;-
```

```
      3DF6   960 DOUBLE_TO_BYTE: $BAS$MAT_MUL    D, B
      401B   961
```

```
    401B   963 DOUBLE_TO_WORD: $BAS$MAT_MUL    D, W
    4240   964
```

```
4240    966 DOUBLE_TO_LONG: $BAS$MAT_MUL     D, L
4465    967
```

```
     4465   969 DOUBLE_TO_FLOA: $BAS$MAT_MUL    D, F
     468A   970
     468A   972 DOUBLE_TO_DOUBL: $BAS$MAT_MUL   D, D
     4886   973
```

```
4886   975 DOUBLE_TO_GFLOA: $BAS$MAT_MUL   D, G
4AE9   976
```

```
     4AE9   978 DOUBLE_TO_HFLOA: $BAS$MAT_MUL   D, H
```

```
                            4D13  ;+
                            4D13  ; Source1 array is a gfloat array.  Now differentiate on the source2 type.
                            4D13  ;-
                            4D13
05    06    02 A3     8F    4D13  GFLOAT: CASEB   DSC$B_DTYPE(R3), #DSC$K_DTYPE_B, #<DSC$K_DTYPE_D - DSC$K_DTYPE_B>
                  002D'     4D18  1$:     .WORD   GFLOAT_TO_BYTE-1$               ; code for byte dtype
                  0256'     4D1A          .WORD   GFLOAT_TO_WORD-1$               ; code for word dtype
                  047F'     4D1C          .WORD   GFLOAT_TO_LONG-1$               ; code for long dtype
                  B3FD      4D1E          .WORD   ERR_DATTYPERR-1$                ; quad not supported
                  06A8'     4D20          .WORD   GFLOAT_TO_FLOAT-1$              ; code for float dtype
                  08D1'     4D22          .WORD   GFLOAT_TO_DOUBL-1$              ; code for double dtype
                            4D24
                            4D24  ;+
                            4D24  ; G and H floating fall outside the range of the CASEB.
                            4D24  ;-
                            4D24
      1B    02 A3     91    4D24          CMPB    DSC$B_DTYPE(R3), #DSC$K_DTYPE_G
               03     12    4D28          BNEQ    2$
             0AEF     31    4D2A          BRW     GFLOAT_TO_GFLOA
                            4D2D
      1C    02 A3     91    4D2D  2$:     CMPB    DSC$B_DTYPE(R3), #DSC$K_DTYPE_H
               03     12    4D31          BNEQ    3$
             0D11     31    4D33          BRW     GFLOAT_TO_HFLOA
                            4D36
      18    02 A3     91    4D36  3$:     CMPB    DSC$B_DTYPE(R3), #DSC$K_DTYPE_DSC
               06     12    4D3A          BNEQ    4$
      53    04 A3     D0    4D3C          MOVL    4(R3), R3                       ; R3 <-- addr of descriptor
               D1     11    4D40          BRB     GFLOAT                          ; CASE again on dtype in desc
                            4D42
             B3D0     31    4D42  4$:     BRW     ERR_DATTYPERR
                            4D45  ;+
                            4D45  ; Now type of source1 and source2 arrays are known.  Use the macro to
                            4D45  ; generate the code for each case
                            4D45  ;-
```

```
4D45  1015 GFLOAT_TO_BYTE: $BAS$MAT_MUL    G, B
4F6E  1016
```

```
     4F6E  1018 GFLOAT_TO_WORD: $BAS$MAT_MUL    G, W
     5197  1019
     5197  1021 GFLOAT_TO_LONG: $BAS$MAT_MUL     G, L
     53C0  1022
     53C0  1024 GFLOAT_TO_FLOAT:$BAS$MAT_MUL    G, F
     55E9  1025
```

```
55E9  1027 GFLOAT_TO_DOUBL:$BAS$MAT_MUL    G, D
581C  1028
```

```
     581C  1030 GFLOAT_TO_GFLOA:$BAS$MAT_MUL    G, G
     5A47  1031
     5A47  1033 GFLOAT_TO_HFLOA:$BAS$MAT_MUL     G, H
     5C78  1034
```

```
                                  5C78  ;+
                                  5C78  ; Source1 array is an hfloat array.  Now differentiate on the source2 type.
                                  5C78  ;-
                                  5C78
     05    06    02 A3      8F    5C78  HFLOAT: CASEB   DSC$B_DTYPE(R3), #DSC$K_DTYPE_B, #<DSC$K_DTYPE_D - DSC$K_DTYPE_B>
                         002D'    5C7D  1$:     .WORD   HFLOAT_TO_BYTE-1$               ; code for byte dtype
                         0256'    5C7F          .WORD   HFLOAT_TO_WORD-1$               ; code for word dtype
                         047F'    5C81          .WORD   HFLOAT_TO_LONG-1$               ; code for long dtype
                         A49R     5C83          .WORD   ERR_DATTYPERR-1$                ; quad not supported
                         06A8'    5C85          .WORD   HFLOAT_TO_FLOAT-1$              ; code for float dtype
                         08D1'    5C87          .WORD   HFLOAT_TO_DOUBL-1$              ; code for double dtype
                                  5C89
                                  5C89  ;+
                                  5C89  ; G and H floating fall outside the range of the CASEB.
                                  5C89  ;-
                                  5C89
           1B    02 A3      91    5C89          CMPB    DSC$B_DTYPE(R3), #DSC$K_DTYPE_G
                    03      12    5C8D          BNEQ    2$
                  0AE5      31    5C8F          BRW     HFLOAT_TO_GFLOA
                                  5C92
           1C    02 A3      91    5C92  2$:     CMPB    DSC$B_DTYPE(R3), #DSC$K_DTYPE_H
                    03      12    5C96          BNEQ    3$
                  0D0B      31    5C98          ERW     HFLOAT_TO_HFLOA
                                  5C9B
           18    02 A3      91    5C9B  3$:     CMPB    DSC$B_DTYPE(R3), #DSC$K_DTYPE_DSC
                    06      12    5C9F          BNEQ    4$
           53    04 A3      D0    5CA1          MOVL    4(R3), R3                       ; R3 <-- addr of descriptor
                    D1      11    5CA5          BRB     HFLOAT                          ; CASE again on dtype in desc
                                  5CA7
                  A46B      31    5CA7  4$:     BRW     ERR_DATTYPERR
                                  5CAA  ;+
                                  5CAA  ; Now type of source1 and source2 arrays are known.  Use the macro to
                                  5CAA  ; generate the code for each case
                                  5CAA  ;-
     5CAA  1071 HFLOAT_TO_BYTE: $BAS$MAT_MUL     H, B
     5ED3  1072
```

```
5ED3  1074 HFLOAT_TO_WORD: $BAS$MAT_MUL    H, W
60FC  1075
```

```
     60FC  1077 HFLOAT_TO_LONG: $BAS$MAT_MUL    H, L
     6325  1078
     6325  1080 HFLOAT_TO_FLOAT:$BAS$MAT_MUL    H, F
     654E  1081
```

```
654E  1083 HFLOAT_TO_DOUBL:$BAS$MAT_MUL    H, D
6777  1084
```

```
6777  1086 HFLOAT_TO_GFLOA:$BAS$MAT_MUL     H, G
69A6  1087
```

```
      69A6  1089 HFLOAT_TO_HFLOA:$BAS$MAT_MUL    H, H
```

```
                         6BD1   1091 ;+
                         6BD1   1092 ; Add has been in byte.  Determine destination type to convert to dest.
                         6BD1   1093 ;-
                         6BD1   1094
                         6BD1   1095 DEST_CASE_B:
      50   1C A5    90   6BD1   1096          MOVB     current_sum(R5), R0                  ; get # to store in R0
         56   5A    D0   6BD5   1097          MOVL     R10, R6                              ; save original pointer
  05    06   02 A6    8F   6BD8   1098 5$:      CASEB    DSC$B_DTYPE(R6), #DSC$K_DTYPE_B, #<DSC$K_DTYPE_D - DSC$K_DTYPE_B>
                  01A7'  6BDD   1099 1$:      .WORD    STORE_BYTE-1$                        ; no conversion needed
                  0288'  6BDF   1100          .WORD    DEST_B_TO_W-1$                       ; code for word dtype
                  038B'  6BE1   1101          .WORD    DEST_B_TO_L-1$                       ; code for long dtype
                  9538   6BE3   1102          .WORD    ERR_DATTYPERR-1$                     ; quad not supported
                  048E'  6BE5   1103          .WORD    DEST_B_TO_F-1$                       ; code for float dtype
                  058D'  6BE7   1104          .WORD    DEST_B_TO_D-1$                       ; code for double dtype
                         6BE9   1105
                         6BE9   1106 ;+
                         6BE9   1107 ; G and H floating fall outside the range of the CASEB.
                         6BE9   1108 ;-
                         6BE9   1109
      1B   02 A6    91   6BE9   1110          CMPB     DSC$B_DTYPE(R6), #DSC$K_DTYPE_G
              03    12   6BED   1111          BNEQ     2$
            06CA    31   6BEF   1112          BRW      DEST_B_TO_G
                         6BF2   1113
      1C   02 A6    91   6BF2   1114 2$:      CMPB     DSC$B_DTYPE(R6), #DSC$K_DTYPE_H
              03    12   6BF6   1115          BNEQ     3$
            07DA    31   6BF8   1116          BRW      DEST_B_TO_H
                         6BFB   1117
      18   02 A6    91   6BFB   1118 3$:      CMPB     DSC$B_DTYPE(R6), #DSC$K_DTYPE_DSC
              06    12   6BFF   1119          BNEQ     4$
      56   04 A6    D0   6C01   1120          MOVL     4(R6), R6                            ; R6 <-- addr of descriptor
              D1    11   6C05   1121          BRB      5$                                   ; CASE again for dtype in desc
                         6C07   1122
            950B    31   6C07   1123 4$:      BRW      ERR_DATTYPERR
                         6C0A   1124
                         6C0A   1125 ;+
                         6C0A   1126 ;+
                         6C0A   1127 ; Add has been in word.  Determine destination type to convert to dest.
                         6C0A   1128 ;-
                         6C0A   1129
                         6C0A   1130 DEST_CASE_W:
      50   1C A5    B0   6C0A   1131          MOVW     current_sum(R5), R0                  ; get # to store in R0
         56   5A    D0   6C0E   1132          MOVL     R10, R6                              ; save original pointer
  05    06   02 A6    8F   6C11   1133 5$:      CASEB    DSC$B_DTYPE(R6), #DSC$K_DTYPE_B, #<DSC$K_DTYPE_D - DSC$K_DTYPE_B>
                  0150'  6C16   1134 1$:      .WORD    DEST_W_TO_B-1$                       ; code for byte dtype
                  0271'  6C18   1135          .WORD    STORE_WORD-1$                        ; no conversion needed
                  0357'  6C1A   1136          .WORD    DEST_W_TO_L-1$                       ; code for long dtype
                  94FF   6C1C   1137          .WORD    ERR_DATTYPERR-1$                     ; quad not supported
                  045A'  6C1E   1138          .WORD    DEST_W_TO_F-1$                       ; code for float dtype
                  055D'  6C20   1139          .WORD    DEST_W_TO_D-1$                       ; code for double dtype
                         6C22   1140
                         6C22   1141 ;+
                         6C22   1142 ; G and H floating fall outside the range of the CASEB.
                         6C22   1143 ;-
                         6C22   1144
      1B   02 A6    91   6C22   1145          CMPB     DSC$B_DTYPE(R6), #DSC$K_DTYPE_G
              03    12   6C26   1146          BNEQ     2$
            0697    31   6C28   1147          BRW      DEST_W_TO_G
```

```
                      6C2B
      1C  02 A6   91  6C2B   2$:      CMPB    DSC$B_DTYPE(R6), #DSC$K_DTYPE_H
             03   12  6C2F            BNEQ    3$
           07A7   31  6C31            BRW     DEST_W_TO_H
                      6C34
      18  02 A6   91  6C34   3$:      CMPB    DSC$B_DTYPE(R6), #DSC$K_DTYPE_DSC
             06   12  6C38            BNEQ    4$
      56  04 A6   D0  6C3A            MOVL    4(R6), R6                       ; R6 <-- addr of descriptor
             D1   11  6C3E            BRB     5$                              ; CASE again for dtype in desc
                      6C40
           94D2   31  6C40   4$:      BRW     ERR_DATTYPERR
                      6C43
                      6C43   ;+
                      6C43   ; Add has been in long.  Determine destination type to convert to dest.
                      6C43   ;-
                      6C43
                      6C43   DEST_CASE_L:
      50  1C A5   D0  6C43            MOVL    current_sum(R5), R0             ; get # to store in R0
         56   5A  D0  6C47            MOVL    R10, R6                         ; save original pointer
  05  06  02 A6   8F  6C4A   5$:      CASEB   DSC$B_DTYPE(R6), #DSC$K_DTYPE_B, #<DSC$K_DTYPE_D - DSC$K_DTYPE_B>
          011C'       6C4F   1$:      .WORD   DEST_L_TO_B-1$                  ; code for byte dtype
          021B'       6C51            .WORD   DEST_L_TO_W-1$                  ; code for word dtype
          033B'       6C53            .WORD   STORE_LONG-1$                   ; no conversion needed
          94C6        6C55            .WORD   ERR_DATTYPERR-1$                ; quad not supported
          0426'       6C57            .WORD   DEST_L_TO_F-1$                  ; code for float dtype
          052D'       6C59            .WORD   DEST_L_TO_D-1$                  ; code for double dtype
                      6C5B   ;+
                      6C5B   ; G and H floating fall outside the range of the CASEB.
                      6C5B   ;-
                      6C5B
      1B  02 A6   91  6C5B            CMPB    DSC$B_DTYPE(R6), #DSC$K_DTYPE_G
             03   12  6C5F            BNEQ    2$
           0664   31  6C61            BRW     DEST_L_TO_G
                      6C64
      1C  02 A6   91  6C64   2$:      CMPB    DSC$B_DTYPE(R6), #DSC$K_DTYPE_H
             03   12  6C68            BNEQ    3$
           0774   31  6C6A            BRW     DEST_L_TO_H
                      6C6D
      18  02 A6   91  6C6D   3$:      CMPB    DSC$B_DTYPE(R6), #DSC$K_DTYPE_DSC
             06   12  6C71            BNEQ    4$
      56  04 A6   D0  6C73            MOVL    4(R6), R6                       ; R6 <-- addr of descriptor
             D1   11  6C77            BRB     5$                              ; CASE again for dtype in desc
                      6C79
           9499   31  6C79   4$:      BRW     ERR_DATTYPERR
                      6C7C
                      6C7C
                      6C7C   ;+
                      6C7C   ; Add has been in float.  Determine destination type to convert to dest.
                      6C7C   ;-
                      6C7C
                      6C7C   DEST_CASE_F:
      50  1C A5   50  6C7C            MOVF    current_sum(R5), R0             ; get # to store in R0
         56   5A  D0  6C80            MOVL    R10, R6                         ; save original pointer
  05  06  02 A6   8F  6C83   5$:      CASEB   DSC$B_DTYPE(R6), #DSC$K_DTYPE_B, #<DSC$K_DTYPE_D - DSC$K_DTYPE_B>
          00E8'       6C88   1$:      .WORD   DEST_F_TO_B-1$                  ; code for byte dtype
          01E7'       6C8A            .WORD   DEST_F_TO_W-1$                  ; code for word dtype
          02EA'       6C8C            .WORD   DEST_F_TO_L-1$                  ; code for long dtype
```

```
                   948D  6C8E             .WORD    ERR_DATTYPERR-1$                  ; quad not supported
                  0401'  6C90             .WORD    STORE_FLOAT-1$                    ; no conversion needed
                  04FD'  6C92             .WORD    DEST_F_TO_D-1$                    ; code for double dtype
                         6C94       ;+
                         6C94       ; G and H floating fall outside the range of the CASEB.
                         6C94       ;-
                         6C94
        1B   02 A6   91  6C94             CMPB     DSC$B_DTYPE(R6), #DSC$K_DTYPE_G
                03   12  6C98             BNEQ     2$
              0631   31  6C9A             BRW      DEST_F_TO_G
                         6C9D
        1C   02 A6   91  6C9D       2$:   CMPB     DSC$B_DTYPE(R6), #DSC$K_DTYPE_H
                03   12  6CA1             BNEQ     3$
              0741   31  6CA3             BRW      DEST_F_TO_H
                         6CA6
        18   02 A6   91  6CA6       3$:   CMPB     DSC$B_DTYPE(R6), #DSC$K_DTYPE_DSC
                06   12  6CAA             BNEQ     4$
        56   04 A6   D0  6CAC             MOVL     4(R6), R6                         ; R6 <-- addr of descriptor
                D1   11  6CB0             BRB      5$                                ; CASE again for dtype in desc
                         6CB2
              9460   31  6CB2       4$:   BRW      ERR_DATTYPERR
                         6CB5
                         6CB5
                         6CB5       ;+
                         6CB5       ; Add has been in double.  Determine destination type to convert to dest.
                         6CB5       ;-
                         6CB5
                         6CB5       DEST_CASE_D:
        50   1C A5   70  6CB5             MOVD     current_sum(R5), R0               ; get # to store in R0&R1
        5A   0C AC   D0  6CB9             MOVL     dest_matrix(AP), R10              ; point to dest matrix
           56   5A   D0  6CBD             MOVL     R10, R6                           ; save original pointer
   05   06   02 A6   8F  6CC0       5$:   CASEB    DSC$B_DTYPE(R6), #DSC$K_DTYPE_B, #<DSC$K_DTYPE_D - DSC$K_DTYPE_B>
                  00B0'  6CC5       1$:   .WORD    DEST_D_TO_B-1$                    ; code for byte dtype
                  01AF'  6CC7             .WORD    DEST_D_TO_W-1$                    ; code for word dtype
                  02B2'  6CC9             .WORD    DEST_D_TO_L-1$                    ; code for long dtype
                   9450  6CCB             .WORD    ERR_DATTYPERR-1$                  ; quad not supported
                  03B5'  6CCD             .WORD    DEST_D_TO_F-1$                    ; code for float dtype
                  0516'  6CCF             .WORD    STORE_DOUBLE-1$                   ; no conversion needed
                         6CD1       ;+
                         6CD1       ; G and H floating fall outside the range of the CASEB.
                         6CD1       ;-
                         6CD1
        1B   02 A6   91  6CD1             CMPB     DSC$B_DTYPE(R6), #DSC$K_DTYPE_G
                03   12  6CD5             BNEQ     2$
              05FA   31  6CD7             BRW      DEST_D_TO_G
                         6CDA
        1C   02 A6   91  6CDA       2$:   CMPB     DSC$B_DTYPE(R6), #DSC$K_DTYPE_H
                03   12  6CDE             BNEQ     3$
              070A   31  6CE0             BRW      DEST_D_TO_H
                         6CE3
        18   02 A6   91  6CE3       3$:   CMPB     DSC$B_DTYPE(R6), #DSC$K_DTYPE_DSC
                06   12  6CE7             BNEQ     4$
        56   04 A6   D0  6CE9             MOVL     4(R6), R6                         ; R6 <-- addr of descriptor
                D1   11  6CED             BRB      5$                                ; CASE again for dtype in desc
                         6CEF
              9423   31  6CEF       4$:   BRW      ERR_DATTYPERR
                         6CF2
```

```
                    6CF2
                    6CF2   ;+
                    6CF2   ; Add has been in gfloat.  Determine destination type to convert to dest.
                    6CF2   ;-
                    6CF2
                    6CF2   DEST_CASE_G:
50    1C A5 50FD    6CF2                MOVG    current_sum(R5), R0                     ; get # to store in R0
   56    5A   D0    6CF7                MOVL    R10, R6                                 ; save original pointer
05    06 02 A6   8F 6CFA   5$:          CASEB   DSC$B_DTYPE(R6), #DSC$K_DTYPE_B, #<DSC$K_DTYPE_D - DSC$K_DTYPE_B>
           007B'    6CFF   1$:          .WORD   DEST_G_TO_B-1$                          ; code for byte dtype
           017E'    6D01                .WORD   DEST_G_TO_W-1$                          ; code for word dtype
           0281'    6D03                .WORD   DEST_G_TO_L-1$                          ; code for long dtype
           9416     6D05                .WORD   ERR_DATTYPERR-1$                        ; quad not supported
           0380'    6D07                .WORD   DEST_G_TO_F-1$                          ; code for float dtype
           049B'    6D09                .WORD   DEST_G_TO_D-1$                          ; code for double dtype
                    6D0B
                    6D0B   ;+
                    6D0B   ; G and H floating fall outside the range of the CASEB.
                    6D0B   ;-
                    6D0B
   1B    02 A6   91 6D0B                CMPB    DSC$B_DTYPE(R6), #DSC$K_DTYPE_G
            03   12 6D0F                BNEQ    2$
          05DB   31 6D11                BRW     STORE_GFLOAT
                    6D14
   1C    02 A6   91 6D14   2$:          CMPB    DSC$B_DTYPE(R6), #DSC$K_DTYPE_H
            03   12 6D18                BNEQ    3$
          06D6   31 6D1A                BRW     DEST_G_TO_H
                    6D1D
   1B    02 A6   91 6D1D   3$:          CMPB    DSC$B_DTYPE(R6), #DSC$K_DTYPE_DSC
            06   12 6D21                BNEQ    4$
   56    04 A6   D0 6D23                MOVL    4(R6), R6                               ; R6 <-- addr of descriptor
            D1   11 6D27                BRB     5$                                      ; CASE again for dtype in desc
                    6D29
          93E9   31 6D29   4$:          BRW     ERR_DATTYPERR
                    6D2C
                    6D2C   ;+
                    6D2C   ; Add has been in hfloat.  Determine destination type to convert to dest.
                    6D2C   ;-
                    6D2C
                    6D2C   DEST_CASE_H:
50    1C A5 70FD    6D2C                MOVH    current_sum(R5`, R0                     ; get # to store in R0
   56    5A   D0    6D31                MOVL    R10, R6                                 ; save original pointer
05    06 02 A6   8F 6D34   5$:          CASEB   DSC$B_DTYPE(R6), #DSC$K_DTYPE_B, #<DSC$K_DTYPE_D - DSC$K_DTYPE_B>
           0047'    6D39   1$:          .WORD   DEST_H_TO_B-1$                          ; code for byte dtype
           014A'    6D3B                .WORD   DEST_H_TO_W-1$                          ; code for word dtype
           024D'    6D3D                .WORD   DEST_H_TO_L-1$                          ; code for long dtype
           93DC     6D3F                .WORD   ERR_DATTYPERR-1$                        ; quad not supported
           034C'    6D41                .WORD   DEST_H_TO_F-1$                          ; code for float dtype
           048E'    6D43                .WORD   DEST_H_TO_D-1$                          ; code for double dtype
                    6D45
                    6D45   ;+
                    6D45   ; G and H floating fall outside the range of the CASEB.
                    6D45   ;-
                    6D45
   1B    02 A6   91 6D45                CMPB    DSC$B_DTYPE(R6), #DSC$K_DTYPE_G
            03   12 6D49                BNEQ    2$
          059D   31 6D4B                BRW     DEST_H_TO_G
```

```
                        6D4E
1C    02 A6      91     6D4E   2$:      CMPB    DSC$B_DTYPE(R6), #DSC$K_DTYPE_H
         03      12     6D52            BNEQ    3$
       06A0      31     6D54            BRW     STORE_HFLOAT
                        6D57
18    02 A6      91     6D57   3$:      CMPB    DSC$B_DTYPE(R6), #DSC$K_DTYPE_DSC
         06      12     6D5B            BNEQ    4$
56    04 A6      D0     6D5D            MOVL    4(R6), R6                       ; R6 <-- addr of descriptor
         D1      11     6D61            BRB     5$                              ; CASE again for dtype in desc
                        6D63
       93AF      31     6D63   4$:      BRW     ERR_DATTYPERR
                        6D66
                        6D66   DEST_W_TO_B:
   50    50      33     6D66            CVTWB   R0, R0                          ; convert
         19      11     6D69            BRB     STORE_BYTE                      ; go store
                        6D6B
                        6D6B   DEST_L_TO_B:
   50    50      F6     6D6B            CVTLB   R0, R0                          ; convert
         14      11     6D6E            BRB     STORE_BYTE                      ; go store
                        6D70
                        6D70   DEST_F_TO_B:
   50    50      48     6D70            CVTFB   R0, R0                          ; convert
         0F      11     6D73            BRB     STORE_BYTE                      ; go store
                        6D75
                        6D75   DEST_D_TO_B:
   50    50      68     6D75            CVTDB   R0, R0                          ; convert
         0A      11     6D78            BRB     STORE_BYTE                      ; go store
                        6D7A
                        6D7A   DEST_G_TO_B:
   50    50 48FD        6D7A            CVTGB   R0, R0                          ; convert
         04      11     6D7E            BRB     STORE_BYTE                      ; go store
                        6D80
                        6D80   DEST_H_TO_B:
   50    50 68FD        6D80            CVTHB   R0, R0                          ; convert
                        6D84                                                    ; fall into store
                        6D84
                        6D84   STORE_BYTE:
   51    5A      D0     6D84            MOVL    R10, R1                         ; pointer to dest descriptor
52    18 A5      D0     6D87            MOVL    current_i(R5), R2               ; current row
53    14 A5      D0     6D8B            MOVL    current_j(R5), R3               ; current column
4A AE    50      90     6D8F            MOVB    R0, DATA(SP)
                        6D93            STORE   B
                 05     6E64            RSB                                     ; go continue loop
                        6E65
                        6E65   DEST_B_TO_W:
   50    50      99     6E65            CVTBW   R0, R0                          ; convert
         1D      11     6E68            BRB     STORE_WORD                      ; go store
                        6E6A
                        6E6A   DEST_L_TO_W:
   50    50      F7     6E6A            CVTLW   R0, R0                          ; convert
         18      11     6E6D            BRB     STORE_WORD                      ; go store
                        6E6F
                        6E6F   DEST_F_TO_W:
   50    50      49     6E6F            CVTFW   R0, R0                          ; convert
         13      11     6E72            BRB     STORE_WORD                      ; go store
                        6E74
                        6E74   DEST_D_TO_W:
```

```
50   2C A5    66  6E74  1376          DIVD2    scale(R5), R0              ; descale for dest
   50    50    69  6E78  1377          CVTDW    R0, R0                     ; convert to word
         0A    11  6E7B  1378          BRB      STORE_WORD                 ; go store
                   6E7D  1379
                   6E7D  1380  DEST_G_TO_W:
   50    50  49FD  6E7D  1381          CVTGW    R0, R0                     ; convert
         04    11  6E81  1382          BRB      STORE_WORD                 ; go store
                   6E83  1383
                   6E83  1384  DEST_H_TO_W:
   50    50  69FD  6E83  1385          CVTHW    R0, R0                     ; convert
                   6E87  1386                                              ; fall into store
                   6E87  1387  STORE_WORD:
   51    5A    D0  6E87  1388          MOVL     R10, R1                    ; pointer to dest descriptor
52   18 A5    D0  6E8A  1389          MOVL     current_i(R5), R2          ; current row
53   14 A5    D0  6E8E  1390          MOVL     current_j(R5), R3          ; current column
4A AE    50    B0  6E92  1391          MOVW     R0, DATA(SP)
                   6E96  1392          STORE    W                          ; store
               05  6F67  1393          RSB                                 ; go continue loop
                   6F68  1394
                   6F68  1395  DEST_B_TO_L:
   50    50    98  6F68  1396          CVTBL    R0, R0                     ; convert
         1D    11  6F6B  1397          BRB      STORE_LONG                 ; go store
                   6F6D  1398
                   6F6D  1399  DEST_W_TO_L:
   50    50    32  6F6D  1400          CVTWL    R0, R0                     ; convert
         18    11  6F70  1401          BRB      STORE_LONG                 ; go store
                   6F72  1402
                   6F72  1403  DEST_F_TO_L:
   50    50    4A  6F72  1404          CVTFL    R0, R0                     ; convert
         13    11  6F75  1405          BRB      STORE_LONG                 ; go store
                   6F77  1406
                   6F77  1407  DEST_D_TO_L:
50   2C A5    66  6F77  1408          DIVD2    scale(R5), R0              ; descale for dest
   50    50    6A  6F7B  1409          CVTDL    R0, R0                     ; convert
         0A    11  6F7E  1410          BRB      STORE_LONG                 ; go store
                   6F80  1411
                   6F80  1412  DEST_G_TO_L:
   50    50  4AFD  6F80  1413          CVTGL    R0, R0                     ; convert
         04    11  6F84  1414          BRB      STORE_LONG                 ; go store
                   6F86  1415
                   6F86  1416  DEST_H_TO_L:
   50    50  6AFD  6F86  1417          CVTHL    R0, R0                     ; convert
                   6F8A  1418                                              ; fall into store
                   6F8A  1419
                   6F8A  1420  STORE_LONG:
   51    5A    D0  6F8A  1421          MOVL     R10, R1                    ; pointer to dest descriptor
52   18 A5    D0  6F8D  1422          MOVL     current_i(R5), R2          ; current row
53   14 A5    D0  6F91  1423          MOVL     current_j(R5), R3          ; current column
4A AE    50    D0  6F95  1424          MOVL     R0, DATA(SP)
                   6F99  1425          STORE    L                          ; store
               05  706A  1426          RSB                                 ; go continue loop
                   706B  1427
                   706B  1428  DEST_B_TO_F:
   50    50    4C  706B  1429          CVTBF    R0, R0                     ; convert
         19    11  706E  1430          BRB      STORE_FLOAT                ; go store
                   7070  1431
                   7070  1432  DEST_W_TO_F:
```

```
        50    50    4D  7070               CVTWF   R0, R0                  ; convert
              14    11  7073               BRB     STORE_FLOAT             ; go store
                        7075
                        7075   DEST_L_TO_F:
        50    50    4E  7075               CVTLF   R0, R0                  ; convert
              0F    11  7078               BRB     STORE_FLOAT             ; go store
                        707A
                        707A   DEST_D_TO_F:
        50    50    76  707A               CVTDF   R0, R0                  ; convert
              0A    11  707D               BRB     STORE_FLOAT             ; go store
                        707F
                        707F   DEST_G_TO_F:
        50    50  33FD  707F               CVTGF   R0, R0                  ; convert
              04    11  7083               BRB     STORE_FLOAT             ; go store
                        7085
                        7085   DEST_H_TO_F:
        50    50  F6FD  7085               CVTHF   R0, R0                  ; convert
                        7089                                               ; fall into store
                        7089
                        7089   STORE_FLOAT:
        51    5A    D0  7089               MOVL    R10, R1                 ; pointer to dest descriptor
     52   18 A5     D0  708C               MOVL    current_i(R5), R2       ; current row
     53   14 A5     D0  7090               MOVL    current_j(R5), R3       ; current column
     4A AE    50    50  7094               MOVF    R0, DATA(SP)
                        7098               STORE   F                       ; store
                    05  7169               RSB                             ; go continue loop
                        716A
                        716A   DEST_B_TO_D:
        50    50    6C  716A               CVTBD   R0, R0                  ; save double
     50   2C A5     64  716D               MULD2   scale(R5), R0           ; scale for dest
                        7171                                               ;  no integerize neressary
              68    11  7171               BRB     STORE_DOUBLE            ; go store
                        7173
                        7173   DEST_W_TO_D:
        50    50    6D  7173               CVTWD   R0, R0                  ; save double
     50   2C A5     64  7176               MULD2   scale(R5), R0           ; scale for dest
                        717A                                               ;  no integerize necessary
              5F    11  717A               BRB     STORE_DOUBLE            ; go store
                        717C
                        717C   DEST_L_TO_D:
        50    50    6E  717C               CVTLD   R0, R0                  ; save double
     50   2C A5     64  717F               MULD2   scale(R5), R0           ; scale for dest
                        7183                                               ;  no integerize necessary
              56    11  7183               BRB     STORE_DOUBLE            ; go store
                        7185
                        7185   DEST_F_TO_D:
        50    50    56  7185               CVTFD   R0, R0                  ; save double
     50   2C A5     64  7188               MULD2   scale(R5), R0           ; scale for dest
     08   2C A5     71  718C               CMPD    scale(R5), #1           ; is the scale 0?
              06    13  7190               BEQL    1$                      ; yes, do not integerize
   00000000'GF      16  7192               JSB     G^MTH$DINT_R4           ; no, integerize
              41    11  7198   1$:         BRB     STORE_DOUBLE            ; go store
                        719A
                        719A   DEST_G_TO_D:
                        719A               ;+
                        719A               ; Note the intermediate conversion to hfloat.
                        719A               ;-
```

```
         7E     52    D0  719A  1490          MOVL    R2, -(SP)                   ; save regs which CVTGH
         7E     53    D0  719D  1491          MOVL    R3, -(SP)                   ;  will destroy
         50     50  56FD  71A0  1492          CVTGH   R0, R0                      ; cvt gfloat to hfloat
         50     50  F7FD  71A4  1493          CVTHD   R0, R0                      ; cvt to desired double
         53     8E    D0  71A8  1494          MOVL    (SP)+, R3                   ; restore regs
         52     8E    D0  71AB  1495          MOVL    (SP)+, R2
      50     2C A5    64  71AE  1496          MULD2   scale(R5), R0               ; scale for dest
      08     2C A5    71  71B2  1497          CMPD    scale(R5), #1               ; scale 0?
                23    13  71B6  1498          BEQL    STORE_DOUBLE                ; yes, don't integerize
         7E     54    D0  71B8  1499          MOVL    R4, -(SP)                   ; save R4
    00000000'GF       16  71BB  1500          JSB     G^MTH$DINT_R4               ; integerize
         54     8E    D0  71C1  1501          MOVL    (SP)+, R4                   ; restore R4
              0014    31  71C4  1502          BRW     STORE_DOUBLE
                          71C7  1503
                          71C7  1504 DEST_H_TO_D:
         50     50  F7FD  71C7  1505          CVTHD   R0, R0                      ; save double
      50     2C A5    64  71CB  1506          MULD2   scale(R5), R0               ; scale for dest
      08     2C A5    71  71CF  1507          CMPD    scale(R5), #1               ; is the scale 0?
                06    13  71D3  1508          BEQL    STORE_DOUBLE                ; yes, do not integerize
    00000000'GF       16  71D5  1509          JSB     G^MTH$DINT_R4               ; no, integerize
                          71DB  1510                                              ; fall into store
                          71DB  1511 STORE_DOUBLE:
         52     5A    D0  71DB  1512          MOVL    R10, R2                     ; pointer to dest descriptor
      53     18 A5    D0  71DE  1513          MOVL    current_i(R5), R3           ; current row
      54     14 A5    D0  71E2  1514          MOVL    current_j(R5), R4           ; current column
      4A AE     50    70  71E6  1515          MOVD    R0, DATA(SP)
                          71EA  1516          STORE   D                           ; store
                      05  72BB  1517          RSB                                 ; go continue loop
                          72BC  1518
                          72BC  1519 DEST_B_TO_G:
         50     50  4CFD  72BC  1520          CVTBG   R0, R0                      ; convert
                2D    11  72C0  1521          BRB     STORE_GFLOAT                ; go store
                          72C2  1522
                          72C2  1523 DEST_W_TO_G:
         50     50  4DFD  72C2  1524          CVTWG   R0, R0                      ; convert
                27    11  72C6  1525          BRB     STORE_GFLOAT                ; go store
                          72C8  1526
                          72C8  1527 DEST_L_TO_G:
         50     50  4EFD  72C8  1528          CVTLG   R0, R0                      ; convert
                21    11  72CC  1529          BRB     STORE_GFLOAT                ; go store
                          72CE  1530
                          72CE  1531 DEST_F_TO_G:
         50     50  99FD  72CE  1532          CVTFG   R0, R0                      ; convert
                1B    11  72D2  1533          BRB     STORE_GFLOAT                ; go store
                          72D4  1534
                          72D4  1535 DEST_D_TO_G:
                          72D4  1536          ;+
                          72D4  1537          ; Note the intermediate corversion to hfloat.
                          72D4  1538          ;-
         7E     52    D0  72D4  1539          MOVL    R2, -(SP)                   ; save regs which CVTDH
         7E     53    D0  72D7  1540          MOVL    R3, -(SP)                   ;  will destroy
         50     50  32FD  72DA  1541          CVTDH   R0, R0                      ; cvt dbl to hfloat
         50     50  76FD  72DE  1542          CVTHG   R0, R0                      ; cvt to desired gfloat
         53     8E    D0  72E2  1543          MOVL    (SP)+, R3                   ; restore regs
         52     8E    D0  72E5  1544          MOVL    (SP)+, R2
              0004    31  72E8  1545          BRW     STORE_GFLOAT
                          72EB  1546
```

```
                  72EB  1547 DEST_H_TO_G:
      50    50 76FD  72EB  1548         CVTHG   R0, R0                          ; convert
                  72EF  1549                                                 ; fall into store
                  72EF  1550
                  72EF  1551 STORE_GFLOAT:
      52    5A   D0  72EF  1552         MOVL    R10, R2                         ; pointer to dest descriptor
   53   18 A5   D0  72F2  1553         MOVL    current_i(R5), R3               ; current row
   54   14 A5   D0  72F6  1554         MOVL    current_j(R5), R4               ; current column
   4A AE    50 50FD  72FA  1555         MOVG    R0, DATA(SP)
                  72FF  1556         STORE   G
              05  73D4  1557         RSB                                     ; go continue loop
                  73D5  1558
                  73D5  1559 DEST_B_TO_H:
      50    50 6CFD  73D5  1560         CVTBH   R0, R0                          ; convert
            1C   11  73D9  1561         BRB     STORE_HFLOAT                    ; go store
                  73DB  1562
                  73DB  1563 DEST_W_TO_H:
      50    50 6DFD  73DB  1564         CVTWH   R0, R0                          ; convert
            16   11  73DF  1565         BRB     STORE_HFLOAT                    ; go store
                  73E1  1566
                  73E1  1567 DEST_L_TO_H:
      50    50 6EFD  73E1  1568         CVTLH   R0, R0                          ; convert
            10   11  73E5  1569         BRB     STORE_HFLOAT                    ; go store
                  73E7  1570
                  73E7  1571 DEST_F_TO_H:
      50    50 98FD  73E7  1572         CVTFH   R0, R0                          ; convert
            0A   11  73EB  1573         BRB     STORE_HFLOAT                    ; go store
                  73ED  1574
                  73ED  1575 DEST_D_TO_H:
      50    50 32FD  73ED  1576         CVTDH   R0, R0                          ; convert
            04   11  73F1  1577         BRB     STORE_HFLOAT                    ; go store
                  73F3  1578
                  73F3  1579 DEST_G_TO_H:
      50    50 56FD  73F3  1580         CVTGH   R0, R0                          ; convert
                  73F7  1581                                                 ; fall into store
                  73F7  1582
                  73F7  1583 STORE_HFLOAT:
      54    5A   D0  73F7  1584         MOVL    R10, R4                         ; pointer to dest descriptor
   56   14 A5   D0  73FA  1585         MOVL    current_j(R5), R6               ; current column
   55   18 A5   D0  73FE  1586         MOVL    current_i(R5), R5               ; current row
   4A AE    50 70FD  7402  1587         MOVH    R0, DATA(SP)
                  7407  1588         STORE   H
              05  74DC  1589         RSB                                     ; go continue loop
                  74DD  1590         .END
```

```
BAS$$SCALE_R1            ********    X  00      DEST_F_TO_B              00006D70 R    02
BAS$$STOP                ********    X  00      DEST_F_TO_D              00007185 R    02
BAS$FETCH_BFA            ********    X  00      DEST_F_TO_G              000072CE R    02
BAS$FET_FA_B_R8          ********    X  00      DEST_F_TO_H              000073E7 R    02
BAS$FET_FA_D_R8          ********    X  00      DEST_F_TO_L              00006F72 R    02
BAS$FET_FA_F_R8          ********    X  00      DEST_F_TO_W              00006E6F R    02
BAS$FET_FA_G_R8          ********    X  00      DEST_G_TO_B              00006D7A R    02
BAS$FET_FA_H_R8          ********    X  00      DEST_G_TO_D              0000719A R    02
BAS$FET_FA_L_R8          ********    X  00      DEST_G_TO_F              0000707F R    02
BAS$FET_FA_W_R8          ********    X  00      DEST_G_TO_H              000073F3 R    02
BAS$K_ARGDONMAT          ********    X  00      DEST_G_TO_L              00006F80 R    02
BAS$K_ARRMUSSAM          ********    X  00      DEST_G_TO_W              00006E7D R    02
BAS$K_DATTYPERR          ********    X  00      DEST_H_TO_B              00006D80 R    02
BAS$K_ILLOPE             ********    X  00      DEST_H_TO_D              000071C7 R    02
BAS$K_MATDIMERR          ********    X  00      DEST_H_TO_F              00007085 R    02
BAS$MAT_MUL              00000000 RG    02      DEST_H_TO_G              000072EB R    02
BAS$MAT_REDIM            ********    X  00      DEST_H_TO_L              00006F86 R    02
BAS$STORE_BFA            ********    X  00      DEST_H_TO_W              00006E83 R    02
BAS$STO_FA_B_R8          ********    X  00      DEST_L_TO_B              00006D6B R    02
BAS$STO_FA_D_R8          ********    X  00      DEST_L_TO_D              0000717C R    02
BAS$STO_FA_F_R8          ********    X  00      DEST_L_TO_F              00007075 R    02
BAS$STO_FA_G_R8          ********    X  00      DEST_L_TO_G              000072C8 R    02
BAS$STO_FA_H_R8          ********    X  00      DEST_L_TO_H              000073E1 R    02
BAS$STO_FA_L_R8          ********    X  00      DEST_L_TO_W              00006E6A R    02
BAS$STO_FA_W_R8          ********    X  00      DEST_MATRIX            = 0000000C
BYTE                     00000122 R     02      DEST_W_TO_B              00006D66 R    02
BYTE_TO_BYTE             00000154 R     02      DEST_W_TO_D              00007173 R    02
BYTE_TO_DOUBLE           000009D1 R     02      DEST_W_TO_F              00007070 R    02
BYTE_TO_FLOAT            000007B0 R     02      DEST_W_TO_G              000072C2 R    02
BYTE_TO_GFLOAT           00000BF8 R     02      DEST_W_TO_H              000073DB R    02
BYTE_TO_HFLOAT           00000E22 R     02      DEST_W_TO_L              00006F6D R    02
BYTE_TO_LONG             0000058F R     02      DOUBLE                   00003DC4 R    02
BYTE_TO_WORD             00000370 R     02      DOUBLE_TO_BYTE           00003DF6 R    02
CLASS                  = 00000045               DOUBLE_TO_DOUBL          0000468A R    02
CURRENT_I              = 00000018               DOUBLE_TO_FLOA           00004465 R    02
CURRENT_J              = 00000C14               DOUBLE_TO_GFLOA          000048B6 R    02
CURRENT_SUM            = 0000001C               DOUBLE_TO_HFLOA          00004AE9 R    02
DATA                   = 0000004A               DOUBLE_TO_LONG           00004240 R    02
DEST_B_TO_D              0000716A R     02      DOUBLE_TO_WORD           0000401B R    02
DEST_B_TO_F              0000706B R     02      DSC$A_A0               = 00000010
DEST_B_TO_G              000072BC R     02      DSC$A_POINTER          = 00000004
DEST_B_TO_H              000073D5 R     02      DSC$B_AFLAGS           = 0000000A
DEST_B_TO_L              00006F68 R     02      DSC$B_CLASS            = 00000003
DEST_B_TO_W              00006E65 R     02      DSC$B_DIMCT            = 0000000B
DEST_CASE_B              00006BD1 R     02      DSC$B_DTYPE            = 00000002
DEST_CASE_D              00006CB5 R     02      DSC$K_CLASS_A          = 00000004
DEST_CASE_F              00006C7C R     02      DSC$K_CLASS_BFA        = 000000BF
DEST_CASE_G              00006CF2 R     02      DSC$K_DTYPE_B          = 00000006
DEST_CASE_H              00006D2C R     02      DSC$K_DTYPE_D          = 0000000B
DEST_CASE_L              00006C43 R     02      DSC$K_DTYPE_DSC        = 00000018
DEST_CASE_W              00006C0A R     02      DSC$K_DTYPE_G          = 0000001B
DEST_D_TO_B              00006D75 R     02      DSC$K_DTYPE_H          = 0000001C
DEST_D_TO_F              0000707A R     02      DSC$L_BYTEOFF          = FFFFFFF8
DEST_D_TO_G              000072D4 R     02      DSC$L_L1_1             = 00000018
DEST_D_TO_H              000073ED R     02      DSC$L_L1_2             = 0000001C
DEST_D_TO_L              00006F77 R     02      DSC$L_L2_2             = 00000024
DEST_D_TO_W              00006E74 R     02      DSC$L_LOGUNIT          = FFFFFFFC
```

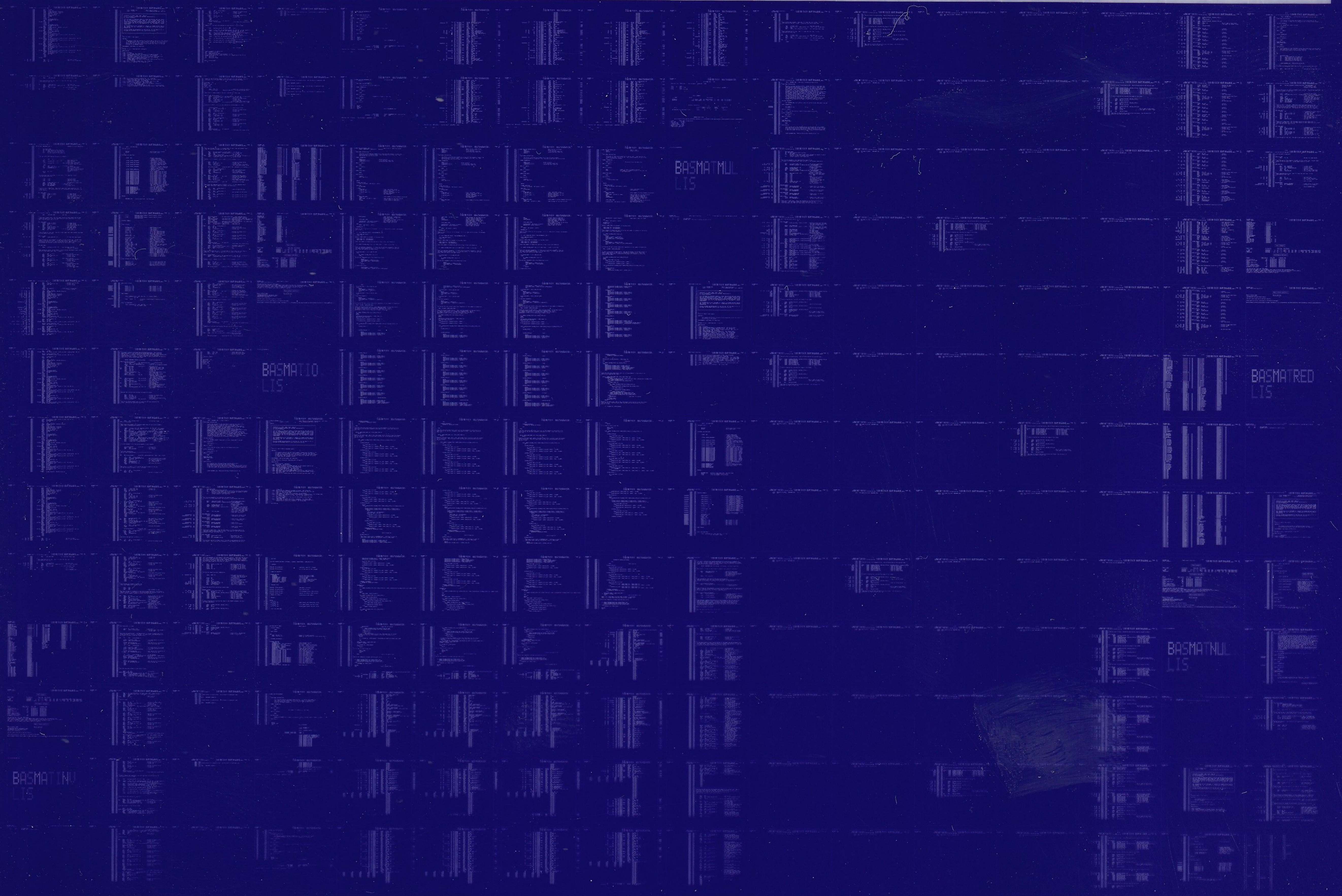
```
DSC$L_M1                         = 00000014                 LOOP_I_DG                        000048C4 R     02
DSC$L_M2                         = 00000018                 LOOP_I_DH                        00004AF7 R     02
DSC$L_U1_1                       = 0000001C                 LOOP_I_DL                        0000424E R     02
DSC$L_U1_2                       = 00000020                 LOOP_I_DW                        00004029 R     02
DSC$L_U2_2                       = 00000028                 LOOP_I_FB                        00002EDE R     02
DSC$V_FL_BOUNDS                  = 00000007                 LOOP_I_FD                        00003757 R     02
DSC$W_LENGTH                     = 00000000                 LOOP_I_FF                        0000353B R     02
DTYPE                            = 00000044                 LOOP_I_FG                        0000397E R     02
ERR_ARGDONMAT                      00000041 R     02        LOOP_I_FH                        00003BA8 R     02
ERR_ARRMUSSAM                      0000004E R     02        LOOP_I_FL                        0000331C R     02
ERR_DATTYPERR                      00000115 R     02        LOOP_I_FW                        000030FD R     02
ERR_ILLOPE                         0000005B R     02        LOOP_I_GB                        00004D53 R     02
ERR_MATDIMERR                      00000034 R     02        LOOP_I_GD                        000055F7 R     02
FLOAT                              00002E9E R     02        LOOP_I_GF                        000053CE R     02
FLOAT_TO_BYTE                      00002ED0 R     02        LOOP_I_GG                        0000582A R     02
FLOAT_TO_DOUBL                     00003749 R     02        LOOP_I_GH                        00005A55 R     02
FLOAT_TO_FLOAT                     0000352D R     02        LOOP_I_GL                        000051A5 R     02
FLOAT_TO_GFLOA                     00003970 R     02        LOOP_I_GW                        00004F7C R     02
FLOAT_TO_HFLOA                     00003B9A R     02        LOOP_I_HB                        00005CB8 R     02
FLOAT_TO_LONG                      0000330E R     02        LOOP_I_HD                        0000655C R     02
FLOAT_TO_WORD                      000030EF R     02        LOOP_I_HF                        00006333 R     02
GFLOAT                             00004D13 R     02        LOOP_I_HG                        00006785 R     02
GFLOAT_TO_BYTE                     00004D45 R     02        LOOP_I_HH                        000069B4 R     02
GFLOAT_TO_DOUBL                    000055E9 R     02        LOOP_I_HL                        0000610A R     02
GFLOAT_TO_FLOAT                    000053C0 R     02        LOOP_I_HW                        00005EE1 R     02
GFLOAT_TO_GFLOA                    0000581C R     02        LOOP_I_LB                        00001FB6 R     02
GFLOAT_TO_HFLOA                    00005A47 R     02        LOOP_I_LD                        00002831 R     02
GFLOAT_TO_LONG                     00005197 R     02        LOOP_I_LF                        00002610 R     02
GFLOAT_TO_WORD                     00004F6E R     02        LOOP_I_LG                        00002A58 R     02
HFLOAT                             00005C78 R     02        LOOP_I_LH                        00002C82 R     02
HFLOAT_TO_BYTE                     00005CAA R     02        LOOP_I_LL                        000023F4 R     02
HFLOAT_TO_DOUBL                    0000654E R     02        LOOP_I_LW                        000021D5 R     02
HFLOAT_TO_FLOAT                    00006325 R     02        LOOP_I_WB                        0000108C R     02
HFLOAT_TO_GFLOA                    00006777 R     02        LOOP_I_WD                        00001909 R     02
HFLOAT_TO_HFLOA                    000069A6 R     02        LOOP_I_WF                        000016E8 R     02
HFLOAT_TO_LONG                     000060FC R     02        LOOP_I_WG                        00001B30 R     02
HFLOAT_TO_WORD                     00005ED3 R     02        LOOP_I_WH                        00001D5A R     02
INIT_STACK                         000000C3 R     02        LOOP_I_WL                        000014C7 R     02
INIT_TWO_SUBS                      00000068 R     02        LOOP_I_WW                        000012AB R     02
LONG                               00001F76 R     02        LOOP_J_BB                        00000167 R     02
LONG_TO_BYTE                       00001FA8 R     02        LOOP_J_BD                        000009E4 R     02
LONG_TO_DOUBLE                     00002823 R     02        LOOP_J_BF                        000007C3 R     02
LONG_TO_FLOAT                      00002602 R     02        LOOP_J_BG                        00000C0B R     02
LONG_TO_GFLOAT                     00002A4A R     02        LOOP_J_BH                        00000E35 R     02
LONG_TO_HFLOAT                     00002C74 R     02        LOOP_J_BL                        000005A2 R     02
LONG_TO_LONG                       000023E6 R     02        LOOP_J_BW                        00000383 R     02
LONG_TO_WORD                       000021C7 R     02        LOOP_J_DB                        00003E09 R     02
LOOP_I_BB                          00000162 R     02        LOOP_J_DD                        0000469D R     02
LOOP_I_BD                          000009DF R     02        LOOP_J_DF                        00004478 R     02
LOOP_I_BF                          000007BE R     02        LOOP_J_DG                        000048C9 R     02
LOOP_I_BG                          00000C06 R     02        LOOP_J_DH                        00004AFC R     02
LOOP_I_BH                          00000E30 R     02        LOOP_J_DL                        00004253 R     02
LOOP_I_BL                          0000059D R     02        LOOP_J_DW                        0000402E R     02
LOOP_I_BW                          0000037E R     02        LOOP_J_FB                        00002EE3 R     02
LOOP_I_DB                          00003E04 R     02        LOOP_J_FD                        0000375C R     02
LOOP_I_DD                          00004698 R     02        LOOP_J_FF                        00003540 R     02
LOOP_I_DF                          00004473 R     02        LOOP_J_FG                        00003983 R     02
```

```
LOOP_J_FH                        00003BAD R      02     LOOP_K_GL                        000051B4 R      02
LOOP_J_FL                        00003321 R      02     LOOP_K_GW                        00004F8B R      02
LOOP_J_FW                        00003102 R      02     LOOP_K_HB                        00005CC7 R      02
LOOP_J_GB                        00004D58 R      02     LOOP_K_HD                        0000656B R      02
LOOP_J_GD                        000055FC R      02     LOOP_K_HF                        00006342 R      02
LOOP_J_GF                        000053D3 R      02     LOOP_K_HG                        00006794 R      02
LOOP_J_GG                        0000582F R      02     LOOP_K_HH                        000069C3 R      02
LOOP_J_GH                        00005A5A R      02     LOOP_K_HL                        00006119 R      02
LOOP_J_GL                        000051AA R      02     LOOP_K_HW                        00005EF0 R      02
LOOP_J_GW                        00004F81 R      02     LOOP_K_LB                        00001FC5 R      02
LOOP_J_HB                        00005CBD R      02     LOOP_K_LD                        00002840 R      02
LOOP_J_HD                        00006561 R      02     LOOP_K_LF                        0000261F R      02
LOOP_J_HF                        00006338 R      02     LOOP_K_LG                        00002A67 R      02
LOOP_J_HG                        0000678A R      02     LOOP_K_LH                        00002C91 R      02
LOOP_J_HH                        000069B9 R      02     LOOP_K_LL                        00002403 R      02
LOOP_J_HL                        0000610F R      02     LOOP_K_LW                        000021E4 R      02
LOOP_J_HW                        00005EE6 R      02     LOOP_K_WB                        0000109B R      02
LOOP_J_LB                        00001FBB R      02     LOOP_K_WD                        00001918 R      02
LOOP_J_LD                        00002836 R      02     LOOP_K_WF                        000016F7 R      02
LOOP_J_LF                        00002615 R      02     LOOP_K_WG                        00001B3F R      02
LOOP_J_LG                        00002A5D R      02     LOOP_K_WH                        00001D69 R      02
LOOP_J_LH                        00002C87 R      02     LOOP_K_WL                        000014D6 R      02
LOOP_J_LL                        000023F9 R      02     LOOP_K_WW                        000012BA R      02
LOOP_J_LW                        000021DA R      02     LOWER_BOUND_J                  = 0000000C
LOOP_J_WB                        00001091 R      02     LOWER_BOUND_K                  = 00000004
LOOP_J_WD                        0000190E R      02     MTH$DINT_R4                      ********    X   00
LOOP_J_WF                        000016ED R      02     POINTER                        = 00000046
LOOP_J_WG                        00001B35 R      02     SCALE                          = 0000002C
LOOP_J_WH                        00001D5F R      02     SEPARATE_DTYPES                  000000E6 R      02
LOOP_J_WL                        000014CC R      02     SF$L_SAVE_FP                   = 0000000C
LOOP_J_WW                        000012B0 R      02     SRC1                           = 00000034
LOOP_K_BB                        00000171 R      02     SRC1_MATRIX                    = 00000004
LOOP_K_BD                        000009EE R      02     SRC2_MATRIX                    = 00000008
LOOP_K_BF                        000007CD R      02     STORE_BYTE                       00006D84 R      02
LOOP_K_BG                        00000C15 R      02     STORE_DOUBLE                     000071DB R      02
LOOP_K_BH                        00000E3F R      02     STORE_FLOAT                      00007089 R      02
LOOP_K_BL                        000005AC R      02     STORE_GFLOAT                     000072EF R      02
LOOP_K_BW                        0000038D R      02     STORE_HFLOAT                     000073F7 R      02
LOOP_K_DB                        00003E13 R      02     STORE_LONG                       00006F8A R      02
LOOP_K_DD                        000046A7 R      02     STORE_WORD                       00006E87 R      02
LOOP_K_DF                        00004482 R      02     STR_LEN                        = 00000042
LOOP_K_DG                        000048D3 R      02     UPPER_BOUND_I                  = 00000010
LOOP_K_DH                        00004806 R      02     UPPER_BOUND_J                  = 00000008
LOOP_K_DL                        0000425D R      02     UPPER_BOUND_K                  = 00000000
LOOP_K_DW                        00004038 R      02     VALUE_DESC                     = 00000042
LOOP_K_FB                        00002EED R      02     VIRTUAL_SAME                     0000009B R      02
LOOP_K_FD                        00003766 R      02     WORD                             0000104C R      02
LOOP_K_FF                        0000354A R      02     WORD_TO_BYTE                     0000107E R      02
LOOP_K_FG                        0000398D R      02     WORD_TO_DOUBLE                   000018FB R      02
LOOP_K_FH                        00003BB7 R      02     WORD_TO_FLOAT                    000016DA R      02
LOOP_K_FL                        0000332B R      02     WORD_TO_GFLOAT                   00001B22 R      02
LOOP_K_FW                        0000310C R      02     WORD_TO_HFLOAT                   00001D4C R      02
LOOP_K_GB                        00004D62 R      02     WORD_TO_LONG                     000014B9 R      02
LOOP_K_GD                        00005606 R      02     WORD_TO_WORD                     0000129D R      02
LOOP_K_GF                        000053DD R      02
LOOP_K_GG                        00005839 R      02
LOOP_K_GH                        00005A64 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                       000074DD  (29917.)  02 (  2.)    PIC    USR    CON    REL    LCL   SHR   EXE   RD   NOWRT NOVEC LONG

                                       +------------------------+
                                       ! Performance indicators !
                                       +------------------------+

Phase                         Page faults    CPU Time       Elapsed Time
-----                         -----------    --------       ------------
Initialization                         28    00:00:00.08    00:00:00.37
Command processing                    105    00:00:00.61    00:00:02.29
Pass 1                               1223    00:00:49.75    00:01:41.26
Symbol table sort                       0    00:00:02.28    00:00:05.05
Pass 2                                872    00:00:12.69    00:00:33.90
Symbol table output                    43    00:00:00.26    00:00:00.59
Psect synopsis output                   4    00:00:00.04    00:00:00.09
Cross-reference output                  0    00:00:00.00    00:00:00.00
Assembler run totals                 2278    00:01:05.71    00:02:23.59

The working set limit was 900 pages.
366875 bytes (717 pages) of virtual memory were used to buffer the intermediate code.
There were 70 pages of symbol table space allocated to hold 479 non-local and 955 local symbols.
1590 source lines were read in Pass 1, producing 91 object records in Pass 2.
46 pages of virtual memory were used to define 11 macros.

                                      +--------------------------+
                                      ! Macro library statistics !
                                      +--------------------------+

Macro library name                           Macros defined
------------------                           --------------
_$255$DUA28:[BASRTL.OBJ]BASRTL.MLB;1                  2
_$255$DUA28:[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$:BASMATMUL/OBJ=OBJ$:BASMATMUL MSRC$:BASMATMUL/UPDATE=(ENH$:BASMATMUL)+LI
```

0026 AH-BT13A-SE
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY



BASMATMUL
LIS

BASMATIO
LIS

BASMATRED
LIS

BASMATNUL
LIS

BASMATINV
LIS