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

```
BBBBBBBB      AAAAAA      SSSSSSSS  MM      MM    AAAAAA    TTTTTTTTTT    AAAAAA      SSSSSSSS    SSSSSSSS
BBBBBBBB      AAAAAA      SSSSSSSS  MM      MM    AAAAAA    TTTTTTTTTT    AAAAAA      SSSSSSSS    SSSSSSSS
BB      BB  AA      AA  SS        MMMM  MMMM  AA      AA      TT      AA      AA  SS          SS
BB      BB  AA      AA  SS        MMMM  MMMM  AA      AA      TT      AA      AA  SS          SS
BB      BB  AA      AA  SS        MM  MM  MM  AA      AA      TT      AA      AA  SS          SS
BB      BB  AA      AA  SS        MM  MM  MM  AA      AA      TT      AA      AA  SS          SS
BBBBBBBB    AA      AA    SSSSSS  MM      MM  AA      AA      TT      AA      AA    SSSSSS      SSSSSS
BBBBBBBB    AA      AA    SSSSSS  MM      MM  AA      AA      TT      AA      AA    SSSSSS      SSSSSS
BB      BB  AAAAAAAAAA          SS  MM      MM  AAAAAAAAAA      TT      AAAAAAAAAA          SS          SS
BB      BB  AAAAAAAAAA          SS  MM      MM  AAAAAAAAAA      TT      AAAAAAAAAA          SS          SS
BB      BB  AA      AA          SS  MM      MM  AA      AA      TT      AA      AA          SS          SS      ....
BB      BB  AA      AA          SS  MM      MM  AA      AA      TT      AA      AA          SS          SS      ....
BBBBBBBB    AA      AA  SSSSSSSS    MM      MM  AA      AA      TT      AA      AA  SSSSSSSS    SSSSSSSS        ....
BBBBBBBB    AA      AA  SSSSSSSS    MM      MM  AA      AA      TT      AA      AA  SSSSSSSS    SSSSSSSS        ....

LL              IIIIII      SSSSSSSS
LL              IIIIII      SSSSSSSS
LL                II      SS
LL                II      SS
LL                II      SS
LL                II        SSSSSS
LL                II        SSSSSS
LL                II              SS
LL                II              SS
LL                II              SS
LL                II              SS
LLLLLLLLLL      IIIIII    SSSSSSSS
LLLLLLLLLL      IIIIII    SSSSSSSS
```

Table of contents

```
0000     1          .TITLE  BAS$MAT_ASSIGN
0000     2          .IDENT  /1-017/                 ; File: BASMATASS.MAR Edit: MDL1017
0000     3 ;
0000     4 ;*****************************************************************************
0000     5 ;*                                                                           *
0000     6 ;*  COPYRIGHT (c) 1978, 1980, 1982, 1984 BY                                  *
0000     7 ;*  DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.                   *
0000     8 ;*  ALL RIGHTS RESERVED.                                                     *
0000     9 ;*                                                                           *
0000    10 ;*  THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED    *
0000    11 ;*  ONLY IN  ACCORDANCE WITH  THE  TERMS  OF  SUCH  LICENSE  AND WITH THE   *
0000    12 ;*  INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR  ANY  OTHER    *
0000    13 ;*  COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY    *
0000    14 ;*  OTHER PERSON.  NO TITLE TO AND OWNERSHIP OF  THE  SOFTWARE IS  HEREBY    *
0000    15 ;*  TRANSFERRED.                                                             *
0000    16 ;*                                                                           *
0000    17 ;*  THE INFORMATION IN THIS SOFTWARE IS  SUBJECT TO CHANGE WITHOUT NOTICE    *
0000    18 ;*  AND  SHOULD  NOT  BE  CONSTRUED AS  A COMMITMENT BY DIGITAL EQUIPMENT   *
0000    19 ;*  CORPORATION.                                                             *
0000    20 ;*                                                                           *
0000    21 ;*  DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE  OR  RELIABILITY OF ITS    *
0000    22 ;*  SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.                  *
0000    23 ;*                                                                           *
0000    24 ;*                                                                           *
0000    25 ;*****************************************************************************
0000    26 ;
0000    27
0000    28 ;++
0000    29 ; FACILITY: BASIC code support
0000    30 ;
0000    31 ; ABSTRACT:
0000    32 ;
0000    33 ;       This module copies one matrix to another.
0000    34 ;
0000    35 ; ENVIRONMENT: User Mode, AST Reentrant
0000    36 ;
0000    37 ;--
0000    38 ; AUTHOR: R. Will, CREATION DATE: 29-May-79
0000    39 ;
0000    40 ; MODIFIED BY:
0000    41 ;++
0000    42 ; 1-001 - Original
0000    43 ; 1-002 - Add support for byte, g and h floating.  PLL 15-Sep-81
0000    44 ; 1-003 - More modifications for new data types.  PLL 24-Sep-81
0000    45 ; 1-004 - Change external references to G^ RNH 25-Sep-81
0000    46 ; 1-005 - Substitute a macro for the calls to the array fetch and store
0000    47 ;         routines.  This should speed things up.  PLL 6-Nov-81
0000    48 ; 1-006 - STORE macro must handle g & h floating.  PLL 11-Nov-81
0000    49 ; 1-007 - Correct a run-time expression FETCH and STORE macros.  PLL 20-Jan-82
0000    50 ; 1-008 - Correct FETCH, STORE again.  PLL 23-Feb-82
0000    51 ; 1-009 - Don't list macro expansions.  PLL 16-Mar-82
0000    52 ; 1-010 - Fix branch instruction which excluded dynamic strings.  PLL 26-Mar-1982
0000    53 ; 1-011 - Remove FETCH and STORE macros; they are now located in macro
0000    54 ;         library MATRIXMAC.OLB.  LB 19-May-1982
0000    55 ; 1-012 - Add support for dynamically mapped arrays.  PLL 1-Jul-1982
0000    56 ; 1-013 - Added code to load up DATA before calling macro STORE.  Also
0000    57 ;         changed CASE stmt code that was enhanced to use R8 instead
```

```
0000     58 ;          of R11.  LEB 4-JUL-1982.
0000     59 ; 1-014 - Changed own storage to stack storage.  PLL 9-Jul-1982
0000     60 ; 1-015 - in D->G and G->D assignments, after promoting to Hfloat, demote
0000     61 ;          back down to destination datatype.  MDL 15-Oct-1982
0000     62 ; 1-016 - Use G^ for ALL externals.  SBL 16-Nov-1982
0000     63 ; 1-017 - there are new externals with the BASRTL/BASMATRTL breakup.  Make
0000     64 ;          them G^ too.  MDL 26-May-1983
0000     65 ;--
```

```
0000     67          .SBTTL  DECLARATIONS
0000     68 ;
0000     69 ; INCLUDE FILES:
0000     70 ;
0000     71
0000     72          $DSCDEF                         ; define descriptor offsets
0000     73          $SFDEF                          ; use to get scale
0000     74
0000     75 ;
0000     76 ; EXTERNAL DECLARATIONS:
0000     77 ;
0000     78
0000     79          .DSABL  GBL                     ; Prevent undeclared
0000     80                                          ;  symbols from being
0000     81                                          ;  automatically global.
0000     82          .EXTRN  BAS$K_ARGDONMAT         ; signalled if all 3 blocks
0000     83                                          ;  not present in array desc
0000     84                                          ;  or dimct = 0
0000     85          .EXTRN  BAS$K_DATTYPERR         ; signalled if dtype of array
0000     86                                          ; isn't word long float double
0000     87                                          ; or if not string to string
0000     88          .EXTRN  BAS$STO_FA_W_R8         ; array element store for word
0000     89          .EXTRN  BAS$STO_FA_L_R8         ; array element store for long
0000     90          .EXTRN  BAS$STO_FA_F_R8         ; array element store - float
0000     91          .EXTRN  BAS$STO_FA_D_R8         ; array element store - double
0000     92          .EXTRN  BAS$FET_FA_W_R8         ; array element fetch - word
0000     93          .EXTRN  BAS$FET_FA_L_R8         ; array element fetch - long
0000     94          .EXTRN  BAS$FET_FA_F_R8         ; array element fetch - float
0000     95          .EXTRN  BAS$FET_FA_D_R8         ; array element fetch - double
0000     96          .EXTRN  BAS$STO_FA_B_R8         ; array element store - byte
0000     97          .EXTRN  BAS$STO_FA_G_R8         ; array element store - gfloat
0000     98          .EXTRN  BAS$STO_FA_H_R8         ; array element store - hfloat
0000     99          .EXTRN  BAS$FET_FA_B_R8         ; array element fetch - byte
0000    100          .EXTRN  BAS$FET_FA_G_R8         ; array element fetch - gfloat
0000    101          .EXTRN  BAS$FET_FA_H_R8         ; array element fetch - hfloat
0000    102          .EXTRN  BAS$STORE_BFA           ; array element store any type
0000    103          .EXTRN  BAS$FETCH_BFA           ; array element fetch any type
0000    104          .EXTRN  BAS$MAT_REDIM           ; check if redimensioning of
0000    105                                          ;  dest array is necessary, if
0000    106                                          ;  so, do it
0000    107          .EXTRN  BAS$$SCALE_R1           ; scale the double procision
0000    108          .EXTRN  STR$FREE1_DX_R4         ; free a string
0000    109          .EXTRN  BAS$$STOP               ; signal fatal errors
0000    110          .EXTRN  MTH$DINT_R4             ; truncate dbl precision number
0000    111
0000    112 ;
0000    113 ; MACROS:
0000    114 ;
0000    115
0000    116 ;        $BAS$MAT_ASS    copy loop algorithm,  see next page
0000    117 ;        FETCH           fetch an element from an array (found in macro library
0000    118 ;                        MATRIXMAC.OLB)
0000    119 ;        STORE           store an element into an array (found in macro library
0000    120 ;                        MATRIXMAC.OLB)
0000    121
0000    122 ;
0000    123 ; EQUATED SYMBOLS:
```

```
                     0000   124 ;
                     0000   125
00000004             0000   126          first_arg = 4                   ; arg offset for str copy
00000008             0000   127          second_arg = 8                  ; arg offset for str copy
0000000C             0000   128          index1 = 12                     ; stack offset for str copy
00000010             0000   129          index2 = 16                     ; stack offset for str copy
00000014             0000   130          temp_desc = 20                  ; stack offset for str copy
00000000             0000   131          lower_bnd2 = 0                  ; stack offset for temp
00000004             0000   132          lower_bnd1 = 4                  ; stack offset for temp
00000008             0000   133          upper_bnd1 = 8                  ; stack offset for temp
0000000C             0000   134          value_desc = 12                 ; output descriptor
0000000C             0000   135          str_len = 12                    ; length field in descriptor
0000000E             0000   136          dtype = 14                      ; data type field in desc
0000000F             0000   137          class = 15                      ; class field within desc
00000010             0000   138          pointer = 16                    ; pointer field within desc
00000014             0000   139          data = 20                       ; DATA (for 4 longwords)
00000018             0000   140          dsc$l_l1_1 = 24                 ; desc offset if 1 sub
0000001C             0000   141          dsc$l_u1_1 = 28                 ; desc offset if 1 sub
0000001C             0000   142          dsc$l_l1_2 = 28                 ; desc offset if 2 sub
00000020             0000   143          dsc$l_u1_2 = 32                 ; desc offset if 2 sub
00000024             0000   144          dsc$l_l2_2 = 36                 ; desc offset if 2 sub
00000028             0000   145          dsc$l_u2_2 = 40                 ; desc offset if 2 sub
                     0000   146
                     0000   147 ;
                     0000   148 ; OWN STORAGE:
                     0000   149 ;
                     0000   150
                     0000   151 ;          NONE
                     0000   152
                     0000   153 ;
                     0000   154 ; PSECT DECLARATIONS:
                     0000   155 ;
                00000000    156          .PSECT _BAS$CODE PIC, USR, CON, REL, LCL, SHR, -
                     0000   157                       EXE, RD, NOWRT, LONG
                     0000   158
```

```
0000    160 ;+
0000    161 ; This macro contains the looping mechanism for accessing all elements of
0000    162 ; an array.  It also contains all the logic for all the combinations of data
0000    163 ; types and scaling.  A macro is used to make it easy to maintain the parallel
0000    164 ; code for all the different data types.
0000    165 ;-
0000    166         .MACRO  $BAS$MAT_ASS src_dtype, dest_dtype ; copy algorithm
0000    167
0000    168 ;+
0000    169 ; Loop through all the rows.  Row and column upper and lower bounds have been
0000    170 ; initialized on the stack.
0000    171 ;-
0000    172
0000    173 LOOP_1ST_SUB'src_dtype'dest_dtype':
0000    174         MOVL    lower_bnd2(SP), R11             ; R11 has 2nd lower bound
0000    175
0000    176 ;+
0000    177 ; Loop through all the elements (columns) of the current row.  Column lower
0000    178 ; bound is initialized in R11.  Column upper bound is on the stack.
0000    179 ; Distinguish array by data type so that the correct fetch routine can
0000    180 ; retrieve the data, the correct conversion can be done and the correct
0000    181 ; store routine can be called.
0000    182 ;-
0000    183
0000    184 LOOP_2ND_SUB'src_dtype'dest_dtype':
0000    185
0000    186 ;+
0000    187 ; Get the data from source array
0000    188 ;-
0000    189
0000    190         MOVL    R10, R0                         ; pointer to array dest
0000    191         MOVL    lower_bnd1(SP), R1              ; current  ow
0000    192         MOVL    R11, R2                         ; current col
0000    193         FETCH   'src_dtype'                     ; fetch data from src array
0000    194
0000    195 ;+
0000    196 ; If the data types of the source and destination arrays are different,
0000    197 ; convert the data to the destination type.  If scaling is needed (ie if
0000    198 ; at least one but not both of the arrays is double) scale the data.
0000    199 ;-
0000    200
0000    201         .IF     DIF     src_dtype, dest_dtype   ; src and dest arrays are not
0000    202                                                 ; same data type
0000    203         .IF     IDN     src_dtype, D            ; source is double
0000    204         MOVD    R0, -(SP)                       ; save the data
0000    205         MOVL    SF$L_SAVE_FP(FP), R0            ; pass FP to get scale
0000    206         JSB     G^BAS$$SCALE_R1                 ; get scale in R0 & R1
0000    207                                                 ; call a BLISS routine because
0000    208                                                 ; the frame offsets are only
0000    209                                                 ; defined for BLISS
0000    210         DIVD3   R0, (SP)+, R0                   ; scale
0000    211         .IF     IDN     dest_dtype, G           ; special case dbl to gfloat
0000    212         CVTDH   R0, R0                          ; must be promoted to hfloat
0000    213         CVTHG   R0, R0                          ; then back down to gfloat
0000    214         .IFF
0000    215         CVT'src_dtype'dest_dtype'       R0, R0  ; only if dbl to non-gfloat
0000    216         .ENDC
```

```
0000    217          .IFF
0000    218          .IF      IDN     src_dtype, G            ; special case gfloat to dbl
0000    219          .IF      IDN     dest_dtype, D
0000    220          CVTGH    R0, R0                          ; promote to hfloat
0000    221          CVTHD    R0, R0                          ; then back down to dbl
0000    222          .IFF
0000    223          CVT'src_dtype'dest_dtype'       R0, R0  ; convert data from R0 into R0
0000    224                                                  ;  R0 & R1 for double
0000    225          .ENDC
0000    226          .IFF
0000    227          CVT'src_dtype'dest_dtype'       R0, R0  ; src not D or G - OK to convert
0000    228          .ENDC
0000    229          .ENDC
0000    230          .IF      IDN     dest_dtype, D           ; dest is double
0000    231          .IF      DIF     src_dtype, G            ; and src is not gfloat
0000    232          MOVD     R0, -(SP)                       ; save the data
0000    233          MOVL     SF$L_SAVE_FP(FP), R0            ; pass FP to get scale
0000    234          JSB      G^BAS$$SCALE_R1                 ; get scale in R0 & R1
0000    235                                                  ; call a BLISS routine because
0000    236                                                  ; the frame offsets are only
0000    237                                                  ; defined for BLISS
0000    238          MULD2    (SP)+, R0                       ; scale
0000    239          JSB      G^MTH$DINT_R4                   ; integerize
0000    240          .ENDC
0000    241          .ENDC
0000    242          .ENDC
0000    243
0000    244 ;+
0000    245 ; Now store the data in the destination array.
0000    246 ; Hfloat passed by value takes 4 words, gfloat and double take 2 words,
0000    247 ; and all other dtypes take 1 longword.
0000    248 ;-
0000    249
0000    250          .IF      IDN     dest_dtype, H           ; dtype is hfloat
0000    251          MOVL     dest_matrix(AP), R4             ; pointer to array desc
0000    252          MOVL     lower_bnd1(SP), R5              ; current row
0000    253          MOVL     R11, R6                         ; current column
0000    254          .IFF
0000    255          .IF      IDN     dest_dtype, G           ; dtype is gfloat
0000    256          MOVL     dest_matrix(AP), R2             ; pointer to array desc
0000    257          MOVL     lower_bnd1(SP), R3              ; current row
0000    258          MOVL     R11, R4                         ; current column
0000    259          .IFF
0000    260          .IF      IDN     dest_dtype, D           ; see if dtype is double
0000    261          MOVL     dest_matrix(AP), R2             ; pointer to array desc
0000    262          MOVL     lower_bnd1(SP), R3              ; current row
0000    263          MOVL     R11, R4                         ; current column
0000    264          .IFF
0000    265          MOVL     dest_matrix(AP), R1             ; all other cases here
0000    266          MOVL     lower_bnd1(SP), R2              ; current row
0000    267          MOVL     R11, R3                         ; current column
0000    268          .ENDC
0000    269          .ENDC
0000    270          .ENDC                                   ; code now same for all dtypes
0000    271          MOV'dest_dtype' R0, DATA(SP)            ; load value into DATA
0000    272          STORE    'dest_dtype'                   ; store in array
0000    273          INCL     R11                            ; get next column
```

```
0000    274          CMPL    R11, R9                          ; see if last column done
0000    275          BGTR    5$
0000    276          BRW     LOOP_2ND_SUB'src_dtype'dest_dtype' ; no, continue inner loop
0000    277
0000    278 ;+
0000    279 ; Have completed entire row.  See if it was the last row.  If not,
0000    280 ; continue with next row.
0000    281 ;-
0000    282
0000    283 5$:      INCL    lower_bnd1(SP)                   ; get next row
0000    284          CMPL    lower_bnd1(SP), upper_bnd1(SP)   ; see if last row done
0000    285          BGTR    7$
0000    286          BRW     LOOP_1ST_SUB'src_dtype'dest_dtype' ; no, continue outer loop
0000    287
0000    288 7$:      RET                                      ; yes, finished
0000    289
0000    290          .ENDM
```

```
                     0000   292          .SBTTL  BAS$MAT_ASSIGN  - Copy one matrix to another
                     0000   293 ;++
                     0000   294 ; FUNCTIONAL DESCRIPTION:
                     0000   295 ;
                     0000   296 ;       Copy one matrix to another.  Redimension the output matrix to have the
                     0000   297 ;       same dimensions as the input matrix.  Initialize all the necessary
                     0000   298 ;       looping information on the stack.  Conversions will have to be done
                     0000   299 ;       from the source data type to the destination data type, so divide
                     0000   300 ;       the looping portion according to the data types.  String arrays
                     0000   301 ;       may only be copied to string arrays so no conversion is necessary
                     0000   302 ;       there.
                     0000   303 ;
                     0000   304 ; CALLING SEQUENCE:
                     0000   305 ;
                     0000   306 ;       CALL BASMAT_ASSIGN (src_matrix.rx.da, dest_matrix.wx.da)
                     0000   307 ;
                     0000   308 ; INPUT PARAMETERS:
                     0000   309 ;
00000004             0000   310          src_matrix = 4
                     0000   311 ;
                     0000   312 ; IMPLICIT INPUTS:
                     0000   313 ;
                     0000   314 ;       Scale from the callers frame to scale double precision.
                     0000   315 ;
                     0000   316 ; OUTPUT PARAMETERS:
                     0000   317 ;
00000008             0000   318          dest_matrix = 8
                     0000   319 ;
                     0000   320 ; IMPLICIT OUTPUTS:
                     0000   321 ;
                     0000   322 ;       NONE
                     0000   323 ;
                     0000   324 ; FUNCTION VALUE:
                     0000   325 ; COMPLETION CODES:
                     0000   326 ;
                     0000   327 ;       NONE
                     0000   328 ;
                     0000   329 ; SIDE EFFECTS:
                     0000   330 ;
                     0000   331 ;       This routine calls the redimensioning routine and the array element
                     0000   332 ;       fetch and store routines and therefore may signal any of their errors.
                     0000   333 ;       It may also signal any of the errors listed in the externals section.
                     0000   334 ;       It may also cause the destination array to have different dimensions.
                     0000   335 ;
                     0000   336 ;--
                     0000   337
     4FFC            0000   338          .ENTRY BAS$MAT_ASSIGN, ^M<R2,R3,R4,R5,R6,R7,R8,R9,R10,R11,IV>
                     0002   339
                     0002   340 ;+
                     0002   341 ;       REGISTER USAGE
                     0002   342 ;       R0 - R8 destroyed by store routines
                     0002   343 ;       R9      upper bound for 2nd subscript
                     0002   344 ;       R10     pointer to array descriptor
                     0002   345 ;       R11     current value of 2nd subscript
                     0002   346 ;-
                     0002   347
                     0002   348 ;+
```

```
                     0002   349 ; Put routine arguments into registers for ease of use.
                     0002   350 ; If block 2 of array descriptor (multipliers) is not present then error.
                     0002   351 ;-
                     0002   352
   5A   04 AC   D0   0002   353          MOVL    src_matrix(AP), R10                     ; ptr to array descr in R10
2A 0A AA    07   E1   0006   354          BBC     #DSC$V_FL_BOUNDS, DSC$B_AFLAGS(R10), ERR_ARGDONMAT
                     000B   355                                                          ; exit if block 3 not
                     000B   356                                                          ; present in descriptor
   0E   02 AA   91   000B   357          CMPB    DSC$B_DTYPE(R10), #DSC$K_DTYPE_T        ; see if not numeric
          03   12   000F   358          BNEQ    1$
        00B3   31   0011   359          BRW     STRING                                  ; text
   18   02 AA   91   0014   360 1$:      CMPB    DSC$B_DTYPE(R10), #DSC$K_DTYPE_DSC      ; see if dynamic string
          0D   12   0018   361          BNEQ    2$                                      ; numeric
                     001A   362
                     001A   363 ;+
                     001A   364 ; Array of descriptors.  This may be a dynamically mapped numeric array, or
                     001A   365 ; an array of dynamic strings.  Decide which and branch.
                     001A   366 ;-
                     001A   367
   50   04 AA   D0   001A   368          MOVL    4(R10), R0                              ; get addr of desc
   0E   02 A0   91   001E   369          CMPB    DSC$B_DTYPE(R0), #DSC$K_DTYPE_T         ; text?
          03   12   0022   370          BNEQ    2$                                      ; no - numeric
        00A0   31   0024   371          BRW     STRING                                  ; text
                     0027   372
                     0027   373 ;+
                     0027   374 ; Allocate value_desc and data on the stack.  This applies to both 1 and 2
                     0027   375 ; dimensional cases.
                     0027   376 ;-
                     0027   377
          7E   7C   0027   378 2$:      CLRQ    -(SP)                                   ; allocate data
          7E   7C   0029   379          CLRQ    -(SP)                                   ;  may be hfloat
          7E   7C   002B   380          CLRQ    -(SP)                                   ; allocate value_desc
                     002D   381
                     002D   382 ;+
                     002D   383 ; Set up limits for looping through all elements
                     002D   384 ;-
                     002D   385
   01   0B AA   91   002D   386          CMPB    DSC$B_DIMCT(R10), #1                    ; determine # of subscripts
          0F   13   0031   387          BEQLU   INIT_ONE_SUB                            ; 1 sub, go init
          2C   1A   0033   388          BGTRU   INIT_TWO_SUBS                           ; >=2 subs, go init
                     0035   389                                                          ; 0 subs, fall into error proc
                     0035   390
                     0035   391 ERR_ARGDONMAT:
00000000'8F   DD   0035   392          PUSHL   #BAS$K_ARGDONMAT                        ; signal error, 0 for dimct
00000000'GF   01   FB   003B   393          CALLS   #1, G^BAS$$STOP                         ; or block 2 or 3 absent
                     0042   394
                     0042   395 ;+
                     0042   396 ; There is only 1 subscript.  Redimension the destination array.
                     0042   397 ; Make both upper and lower bound for 2nd
                     0042   398 ; subscript a 1.  A second subscript will be passed to and ignored by the
                     0042   399 ; store routine.  Put bounds for 1st subscript on stack.
                     0042   400 ;-
                     0042   401
                     0042   402 INIT_ONE_SUB:
        1C AA   DD   0042   403          PUSHL   dsc$l_ul_1(R10)                         ; get bound for redim
        08 AC   DD   0045   404          PUSHL   dest_matrix(AP)                         ; pointer to dest array desc
00000000'GF   02   FB   0048   405          CALLS   #2, G^BAS$MAT_REDIM                     ; redimension the dest
```

```
                 1C AA   DD  004F   406          PUSHL   dsc$l_u1_1(R10)                 ; 1st upper bound
                 18 AA   DD  0052   407          PUSHL   dsc$l_l1_1(R10)                 ; 1st lower bound
                    03   14  0055   408          BGTR    1$                              ; not 0 or neg, do 2nd sub
              6E    01   D0  0057   409          MOVL    #1, (SP)                        ; don't alter col 0
                    01   DD  005A   410 1$:      PUSHL   #1                              ; dummy 2nd upper bound
              59    01   D0  005C   411          MOVL    #1, R9                          ; dummy 2nd lower bound
                    27   11  005F   412          BRB     SEPARATE_DTYPES                 ; go loop
                              0061   413
                              0061   414 ;+
                              0061   415 ; There are 2 subscripts.  Check and redimension the destination array if
                              0061   416 ; necessary.  Put the upper bound for both subscripts on the
                              0061   417 ; stack and make sure that the lower bound for both subscripts will start
                              0061   418 ; at 1 (do not alter row or col 0)
                              0061   419 ;-
                              0061   420
                              0061   421 INIT_TWO_SUBS:
                 28 AA   DD  0061   422          PUSHL   dsc$l_u2_2(R10)                 ; 2nd upper bound
                 20 AA   DD  0064   423          PUSHL   dsc$l_u1_2(R10)                 ; 1st upper bound
                 08 AC   DD  0067   424          PUSHL   dest_matrix(AP)                 ; dest array pointer
00000000'GF      03   FB  006A   425          CALLS   #3, G^BAS$MAT_REDIM             ; redimension destination
                 20 AA   DD  0071   426          PUSHL   dsc$l_u1_2(R10)                 ; 1st upper bound
                 1C AA   DD  0074   427          PUSHL   dsc$l_l1_2(R10)                 ; 1st lower bound
                    03   14  0077   428          BGTR    1$                              ; not row 0 or neg, do cols
              6E    01   D0  0079   429          MOVL    #1, (SP)                        ; start with row 1
          59    28 AA   D0  007C   430 1$:      MOVL    dsc$l_u2_2(R10), R9             ; 2nd upper bound
                 24 AA   DD  0080   431          PUSHL   dsc$l_l2_2(R10)                 ; 2nd lower bound
                    03   14  0083   432          BGTR    SEPARATE_DTYPES                 ; not col 0 or neg, go loop
              6E    01   D0  0085   433          MOVL    #1, (SP)                        ; start with col 1
                              0088   434
                              0088   435 ;+
                              0088   436 ; Algorithm now differs according to data types
                              0088   437 ;-
                              0088   438
                              0088   439 SEPARATE_DTYPES:
                    55   5A   D0  0088   440          MOVL    R10, R5
          05    06    02 A5   8F  008B   441 5$:      CASEB   DSC$B_DTYPE(R5), #DSC$K_DTYPE_B, #<DSC$K_DTYPE_D - DSC$K_DTYPE_B>
                       00E7' 0090   442 2$:      .WORD   BYTE-2$                         ; code for byte dtype
                       0E63' 0092   443          .WORD   WORD-2$                         ; code for word dtype
                       1BDF' 0094   444          .WORD   LONG-2$                         ; code for long dtype
                       002A' 0096   445          .WORD   ERR_DATTYPERR-2$                ; quad not supported
                       295B' 0098   446          .WORD   FLOAT-2$                        ; code for float dtype
                       36D7' 009A   447          .WORD   DOUBLE-2$                       ; code for double dtype
                              009C   448
                              009C   449 ;+
                              009C   450 ; To avoid having to specify ERR_DATTYPERR for all the dtypes between
                              009C   451 ; double and gfloat (12 to 26), check for gfloat and hfloat separately.
                              009C   452 ;-
                              009C   453
              1B    02 A5   91  009C   454          CMPB    DSC$B_DTYPE(R5), #DSC$K_DTYPE_G
                       03   12  00A0   455          BNEQ    3$
                     4492   31  00A2   456          BRW     GFLOAT                          ; code for gfloat dtype
                              00A5   457
              1C    02 A5   91  00A5   458 3$:      CMPB    DSC$B_DTYPE(R5), #DSC$K_DTYPE_H
                       03   12  00A9   459          BNEQ    4$
                     5221   31  00AB   460          BRW     HFLOAT                          ; code for hfloat dtype
                              00AE   461
              18    02 A5   91  00AE   462 4$:      CMPB    DSC$B_DTYPE(R5), #DSC$K_DTYPE_DSC
```

```
                    06   12  00B2    463          BNEQ    ERR_DATTYPERR
                55  04 AA   D0  00B4    464          MOVL    4(R10), R5                      ; get addr of desc
                    D1   11  00B8    465          BRB     5$
                             00BA    466
                             00BA    467 ERR_DATTYPERR:
             00000000'8F    DD  00BA    468          PUSHL   #BAS$K_DATTYPERR                ; Signal error, unsupported
         00000000'GF  01    FB  00C0    469          CALLS   #1, G^BAS$$STOP                 ; dtype in array desc
```

```
                     00C7   472 ;+
                     00C7   473 ; Source array is a string array.
                     00C7   474 ;-
                     00C7   475
                     00C7   476 STRING:
                     00C7   477
                     00C7   478 ;+
                     00C7   479 ;       REGISTER USAGE
                     00C7   480 ;       R2      current bound for 1st subscript
                     00C7   481 ;       R3      upper bound for 1st subscript
                     00C7   482 ;       R4      lower bound for 2nd subscript
                     00C7   483 ;       R5      upper bound for 2nd subscript
                     00C7   484 ;       R6      current value of 2nd subscript
                     00C7   485 ;       R9      pointer to destination array descriptor
                     00C7   486 ;       R10     pointer to source array descriptor
                     00C7   487 ;-
                     00C7   488
                     00C7   489 ;+
                     00C7   490 ; Initialize stack for CALLG to element store routine.  Then divide algorithm
                     00C7   491 ; based on number of subscripts.  Since the dtype in the temporary descriptor
                     00C7   492 ; is text the array fetch and store routines will signal an error if
                     00C7   493 ; the dtype of the array is not text of array of descriptors of type text
                     00C7   494 ; and there is no need to do a check for dtype here.
                     00C7   495 ;-
                     00C7   496
 59    08 AC    DO   00C7   497         MOVL    dest_matrix(AP), R9
       7E       D4   00CB   498         CLRL    -(SP)                           ; pointer of descriptor
020E0000 8F       DD   00CD   499         PUSHL   #<<DSC$K_CLASS_D @ 24> + <DSC$K_DTYPE_T @ 16>> ; 1st desc word
                     00D3   500                                                 ; class, type and 0 length
       7E       7C   00D3   501         CLRQ    -(SP)                           ; space for indices
       5A       DD   00D5   502         PUSHL   R10                             ; ptr to dest desc for call
    0C AE       DF   00D7   503         PUSHAL  12(SP)                          ; pointer to NULL descriptor
01  0B AA       91   00DA   504         CMPB    DSC$B_DIMCT(R10), #1            ; determine # of subscripts
       05       13   00DE   505         BEQLU   INIT_ONE_SUB_S                  ; 1 sub, go init
       26       1A   00E0   506         BGTRU   INIT_TWO_SUBS_S                 ; >=2 subs, go init
     FF50       31   00E2   507         BRW     ERR_ARGDONMAT                   ; 0 subs, error
                     00E5   508
                     00E5   509 ;+
                     00E5   510 ; There is only 1 subscript.  Make both upper and lower bound for 2nd
                     00E5   511 ; subscript a 1.  A second subscript will be passed to and ignored by the
                     00E5   512 ; store routine because the argcount in the arglist of the CALL will not
                     00E5   513 ; include the 2nd subscript.  Put bounds for 1st subscript into registers.
                     00E5   514 ;-
                     00E5   515
                     00E5   516 INIT_ONE_SUB_S:
       03       DD   00E5   517         PUSHL   #3                              ; 3 arguments to store routine
                     00E7   518                                                 ;  ignores 2nd index
    1C AA       DD   00E7   519         PUSHL   dsc$l_u1_1(R10)                 ; get bound for redim
       59       DD   00EA   520         PUSHL   R9                              ; pointer to dest array desc
00000000'GF    02   FB   00EC   521         CALLS   #2, G^BAS$MAT_REDIM             ; redimension the dest array
    53  1C AA       DO   00F3   522         MOVL    dsc$l_u1_1(R10), R3             ; 1st upper bound
    52  18 AA       DO   00F7   523         MOVL    dsc$l_l1_1(R10), R2             ; 1st lower bound
       03       14   00FB   524         BGTR    1$                              ; not 0 or neg, init 2nd bound
    52  01       DO   00FD   525         MOVL    #1, R2                          ; don't alter row 0
    54  01       DO   0100   526 1$:     MOVL    #1, R4                          ; set 2nd lower bnd to 1
    55  01       DO   0103   527         MOVL    #1, R5                          ; set 2nd upper bnd to 1
       2B       11   0106   528         BRB     LOOP_1ST_SUB_S                  ; go loop
```

```
                                0108   529
                                0108   530 ;+
                                0108   531 ; There are 2 subscripts.  Put the upper bound for both subscripts in
                                0108   532 ; registers and make sure that the lower bound for both subscripts will start
                                0108   533 ; at 1 (do not alter row or col 0)
                                0108   534 ;-
                                0108   535
                                0108   536 INIT_TWO_SUBS_S:
                          04    DD     0108   537          PUSHL   #4                              ; 4 arguments to store routine
                       28 AA    DD     010A   538          PUSHL   dsc$l_u2_2(R10)                 ; 2nd upper bound for redim
                       20 AA    DD     010D   539          PUSHL   dsc$l_u1_2(R10)                 ; 1st upper bound for redim
                          59    DD     0110   540          PUSHL   R9                              ; ptr to dest array to redim
00000000'GF               03    FB     0112   541          CALLS   #3, G^BAS$MAT_REDIM             ; redim the dest array
                    55    28 AA    D0     0119   542          MOVL    dsc$l_u2_2(R10), R5             ; 2nd upper bound
                    54    24 AA    D0     011D   543          MOVL    dsc$l_l2_2(R10), R4             ; 2nd lower bound
                          03    14     0121   544          BGTR    1$                              ; not col 0 or neg, do cols
                       54    01    D0     0123   545          MOVL    #1, R4                          ; start with col 1
                    53    20 AA    D0     0126   546 1$:       MOVL    dsc$l_u1_2(R10), R3             ; 1st upper bound
                    52    1C AA    D0     012A   547          MOVL    dsc$l_l1_2(R10), R2             ; 1st lower bound
                          03    14     012E   548          BGTR    LOOP_1ST_SUB_S                  ; not row 0 or neg, go loop
                       52    01    D0     0130   549          MOVL    #1, R2                          ; start with row 1
                                0133   550
                                0133   551 ;+
                                0133   552 ; Loop through all the rows.  Row and column upper and lower bounds have been
                                0133   553 ; initialized in registers.
                                0133   554 ;-
                                0133   555
                                0133   556 LOOP_1ST_SUB_S:
                       56    54    D0     0133   557          MOVL    R4, R6                          ; R6 has 1st lower bound
                                0136   558
                                0136   559 ;+
                                0136   560 ; Loop through all the elements (columns) of the current row.  Column lower
                                0136   561 ; bound is initialized in R6.  Column upper bound is in R5.
                                0136   562 ;-
                                0136   563
                                0136   564 LOOP_2ND_SUB_S:
                                0136   565
                                0136   566 ;+
                                0136   567 ; Fetch the string from the source array.
                                0136   568 ;-
                                0136   569
                    10 AE    56    D0     0136   570          MOVL    R6, index2(SP)                  ; current column
                    0C AE    52    D0     013A   571          MOVL    R2, index1(SP)                  ; current row
                    04 AE    5A    D0     013E   572          MOVL    R10, first_arg(SP)              ; source array desc pointer
                08 AE    14 AE    7E     0142   573          MOVAQ   temp_desc(SP), second_arg(SP)   ; desc to copy fetch
00000000'GF               6E    FA     0147   574          CALLG   (SP), G^BAS$FETCH_BFA           ; copy element to temp_desc
                                014E   575
                                014E   576 ;+
                                014E   577 ; Store the string in the destination array.  Note that the indices are
                                014E   578 ; already set from the fetch.
                                014E   579 ;-
                                014E   580
                04 AE    14 AE    7E     014E   581          MOVAQ   temp_desc(SP), first_arg(SP)    ; string to copy to dest
                    08 AE    59    D0     0153   582          MOVL    R9, second_arg(SP)              ; dest array desc pointer
00000000'GF               6E    FA     0157   583          CALLG   (SP), G^BAS$STORE_BFA           ; store in array
                                56    D6     015E   584          INCL    R6                              ; get next column
                             55    56    D1     0160   585          CMPL    R6, R5                          ; see if last column done
```

```
                 D1   15  0163    586          BLEQ    LOOP_2ND_SUB_S                  ; no, continue inner loop
                          0165    587
                          0165    588 ;+
                          0165    589 ; Have completed entire row.  See if it was the last row.  If not,
                          0165    590 ; continue with next row.
                          0165    591 ;-
                          0165    592
                 52   D6  0165    593          INCL    R2                              ; get next row
              53 52   D1  0167    594          CMPL    R2, R3                          ; see if last row done
                 C7   15  016A    595          BLEQ    LOOP_1ST_SUB_S                  ; no, continue outer loop
                          016C    596
           50    14 AE   7E  016C    597          MOVAQ   temp_desc(SP), R0               ; yes, free the temp string
     00000000'GF      16  0170    598          JSB     G^STR$FREE1_DX_R4
                      04  0176    599          RET                                     ; finished
```

```
                     0177   602 ;+
                     0177   603 ; Source array is a byte array.  Now differentiate on the destination type.
                     0177   604 ;-
                     0177   605
     58   08 AC   D0  0177   606 BYTE:   MOVL    dest_matrix(AP), R8                 ; point to dest descriptor
05   06   02 A8   8F  017B   607 5$:     CASEB   DSC$B_DTYPE(R8), #DSC$K_DTYPE_B, #<DSC$K_DTYPE_D - DSC$K_DTYPE_B>
               002D' 0180   608 1$:     .WORD   BYTE_TO_BYTE-1$                     ; code for byte dtype
               020B' 0182   609         .WORD   BYTE_TO_WORD-1$                     ; code for word dtype
               03EC' 0184   610         .WORD   BYTE_TO_LONG-1$                     ; code for long dtype
               FF3A  0186   611         .WORD   ERR_DATTYPERR-1$                    ; quad not supported
               05CD' 0188   612         .WORD   BYTE_TO_FLOAT-1$                    ; code for float dtype
               07AE' 018A   613         .WORD   BYTE_TO_DOUBLE-1$                   ; code for double dtype
                     018C   614
                     018C   615 ;+
                     018C   616 ; Check for g and h floating separately, since their dtypes do not fit into
                     018C   617 ; the CASEB range.
                     018C   618 ;-
     1B   02 A8   91 018C   619         CMPB    DSC$B_DTYPE(R8), #DSC$K_DTYPE_G
             03   12 0190   620         BNEQ    2$
           0990   31 0192   621         BRW     BYTE_TO_GFLOAT                      ; code for gfloat dtype
                     0195   622
     1C   02 A8   91 0195   623 2$:     CMPB    DSC$B_DTYPE(R8), #DSC$K_DTYPE_H
             03   12 0199   624         BNEQ    3$
           0B6E   31 019B   625         BRW     BYTE_TO_HFLOAT                      ; code for hfloat dtype
                     019E   626
     18   02 A8   91 019E   627 3$:     CMPB    DSC$B_DTYPE(R8), #DSC$K_DTYPE_DSC
             06   12 01A2   628         BNEQ    4$
     58   04 A8   D0 01A4   629         MOVL    4(R8), R8                           ; get addr of desc
             D1   11 01A8   630         BRB     5$                                  ; CASE again for dtype in desc
                     01AA   631
         FF0D   31 01AA   632 4$:     BRW     ERR_DATTYPERR
                     01AD   633
                     01AD   634 ;+
                     01AD   635 ; Now type of source and destination arrays are known.  Use the macro to
                     01AD   636 ; generate the code for each case
                     01AD   637 ;-
                     01AD   638
```

```
01AD    640 BYTE_TO_BYTE:   $BAS$MAT_ASS    B, B
038B    641
```

```
038B    643 BYTE_TO_WORD:   $BAS$MAT_ASS    B, W
056C    644
```

```
056C    646 BYTE_TO_LONG:   $BAS$MAT_ASS    B, L
074D    647
```

```
074D    649 BYTE_TO_FLOAT:  $BAS$MAT_ASS    B, F
092E    650
```

```
092E    652 BYTE_TO_DOUBLE: $BAS$MAT_ASS    B, D
0B25    653
```

```
0B25    655 BYTE_TO_GFLOAT: $BAS$MAT_ASS    B, G
0D0C    656
```

```
                ODOC    658 BYTE_TO_HFLOAT: $BAS$MAT_ASS    B, H
```

```
                     0EF3    660 ;+
                     0EF3    661 ; Source array is a word array.  Now differentiate on the destination type.
                     0EF3    662 ;-
                     0EF3    663
        58   08 AC   D0  0EF3    664 WORD:   MOVL    dest_matrix(AP), R8                 ; point to dest descriptor
05      06   02 A8   8F  0EF7    665 5$:     CASEB   DSC$B_DTYPE(R8), #DSC$K_DTYPE_B, #<DSC$K_DTYPE_D - DSC$K_DTYPE_B>
                002D' 0EFC    666 1$:     .WORD   WORD_TO_BYTE-1$                     ; code for byte dtype
                020E' 0EFE    667         .WORD   WORD_TO_WORD-1$                     ; code for word dtype
                03EC' 0F00    668         .WORD   WORD_TO_LONG-1$                     ; code for long dtype
                F1BE  0F02    669         .WORD   ERR_DATTYPERR-1$                    ; quad not supported
                05CD' 0F04    670         .WORD   WORD_TO_FLOAT-1$                    ; code for float dtype
                07AE' 0F06    671         .WORD   WORD_TO_DOUBL-1$                    ; code for double dtype
                     0F08    672 ;+
                     0F08    673 ; Check for g and h floating separately, since their dtypes do not fit into
                     0F08    674 ; the CASEB range.
                     0F08    675 ;-
        1B   02 A8   91  0F08    676         CMPB    DSC$B_DTYPE(R8), #DSC$K_DTYPE_G
                03   12  0F0C    677         BNEQ    2$
                0990 31  0F0E    678         BRW     WORD_TO_GFLOAT                  ; code for gfloat dtype
                     0F11    679
        1C   02 A8   91  0F11    680 2$:     CMPB    DSC$B_DTYPE(R8), #DSC$K_DTYPE_H
                03   12  0F15    681         BNEQ    3$
                0B6E 31  0F17    682         BRW     WORD_TO_HFLOAT                  ; code for hfloat dtype
                     0F1A    683
        18   02 A8   91  0F1A    684 3$:     CMPB    DSC$B_DTYPE(R8), #DSC$K_DTYPE_DSC
                06   12  0F1E    685         BNEQ    4$
        58   04 A8   D0  0F20    686         MOVL    4(R8), R8                       ; get addr of desc
                D1   11  0F24    687         BRB     5$                              ; CASE again for dtype in desc
                     0F26    688
                F191 31  0F26    689 4$:     BRW     ERR_DATTYPERR
                     0F29    690
                     0F29    691
                     0F29    692 ;+
                     0F29    693 ; Now type of source and destination arrays are known.  Use the macro to
                     0F29    694 ; generate the code for each case
                     0F29    695 ;-
                     0F29    696
```

```
                0F29    698 WORD_TO_BYTE:   $BAS$MAT_ASS    W, B
                110A    699
```

```
                110A    701 WORD_TO_WORD:   $BAS$MAT_ASS    W, W
                12E8    702
```

```
12E8    704 WORD_TO_LONG:   $BAS$MAT_ASS    W, L
1409    705
```

```
14C9    707 WORD_TO_FLOAT:  $BAS$MAT_ASS    W, F
16AA    708
```

```
                        .
16AA    710 WORD_TO_DOUBL:  $BAS$MAT_ASS    W, D
18A1    711
```

```
                18A1    713 WORD_TO_GFLOAT: $BAS$MAT_ASS    W, G
                1A88    714
```

```
                1A88    716 WORD_TO_HFLOAT: $BAS$MAT_ASS    W, H
```

```
                        1C6F    718 ;+
                        1C6F    719 ; Source array is a longword array.  Now differentiate on the destination type
                        1C6F    720 ;-
                        1C6F    721
      58    08 AC   D0  1C6F    722 LONG:   MOVL    dest_matrix(AP), R8                 ; point to dest descriptor
05    06    02 A8   8F  1C73    723 5$:     CASEB   DSC$B_DTYPE(R8), #DSC$K_DTYPE_B, #<DSC$K_DTYPE_D - DSC$K_DTYPE_B>
                002D' 1C78    724 1$:     .WORD   LONG_TO_BYTE-1$                     ; code for byte dtype
                020E' 1C7A    725         .WORD   LONG_TO_WORD-1$                     ; code for word dtype
                03EF' 1C7C    726         .WORD   LONG_TO_LONG-1$                     ; code for long dtype
                E442  1C7E    727         .WORD   ERR_DATTYPERR-1$                    ; quad not supported
                05CD' 1C80    728         .WORD   LONG_TO_FLOAT-1$                    ; code for float dtype
                07AE' 1C82    729         .WORD   LONG_TO_DOUBLE-1$                   ; code for double dtype
                      1C84    730
                      1C84    731
                      1C84    732 ;+
                      1C84    733 ; Check for g and h floating separately, since their dtypes do not fit into
                      1C84    734 ; the CASEB range.
                      1C84    735 ;-
      1B    02 A8   91  1C84    736         CMPB    DSC$B_DTYPE(R8), #DSC$K_DTYPE_G
               03   12  1C88    737         BNEQ    2$
             0990   31  1C8A    738         BRW     LONG_TO_GFLOAT                      ; code for gfloat dtype
                      1C8D    739
      1C    02 A8   91  1C8D    740 2$:     CMPB    DSC$B_DTYPE(R8), #DSC$K_DTYPE_H
               03   12  1C91    741         BNEQ    3$
             0B6E   31  1C93    742         BRW     LONG_TO_HFLOAT                      ; code for hfloat dtype
                      1C96    743
      18    02 A8   91  1C96    744 3$:     CMPB    DSC$B_DTYPE(R8), #DSC$K_DTYPE_DSC
               06   12  1C9A    745         BNEQ    4$
      58    04 A8   D0  1C9C    746         MOVL    4(R8), R8                           ; get addr of desc
               D1   11  1CA0    747         BRB     5$
                        1CA2    748
            E415   31  1CA2    749 4$.     BRW     ERR_DATTYPERR
                        1CA5    750
                        1CA5    751 ;+
                        1CA5    752 ; Now type of source and destination arrays are known.  Use the macro to
                        1CA5    753 ; generate the code for each case
                        1CA5    754 ;-
                        1CA5    755
```

```
                1CA5    757 LONG_TO_BYTE:   $BAS$MAT_ASS    L, B
                1E86    758
```

```
                1E86    760 LONG_TO_WORD:   $BAS$MAT_ASS    L, W
                2067    761
```

```
                2067    763 LONG_TO_LONG:   $BAS$MAT_ASS    L, L
                2245    764
```

```
                2245    766 LONG_TO_FLOAT:  $BAS$MAT_ASS    L, F
                2426    767
```

```
                2426    769 LONG_TO_DOUBLE: $BAS$MAT_ASS    L, D
                261D    770
```

```
261D   772 LONG_TO_GFLOAT: $BAS$MAT_ASS    L, G
2804   773
```

```
     2804    775 LONG_TO_HFLOAT: $BAS$MAT_ASS    L, H
```

```
                          29EB    777 ;+
                          29EB    778 ; Source array is a floating array.  Now differentiate on the destination type
                          29EB    779 ;-
                          29EB    780
      58    08 AC    D0   29EB    781 FLOAT:  MOVL    dest_matrix(AP), R8                 ; point to dest descriptor
05    06    02 A8    8F   29EF    782 5$:     CASEB   DSC$B_DTYPE(R8), #DSC$K_DTYPE_B, #<DSC$K_DTYPE_D - DSC$K_DTYPE_B>
                 002D'     29F4    783 1$:     .WORD   FLOAT_TO_BYTE-1$                    ; code for byte dtype
                 020E'     29F6    784         .WORD   FLOAT_TO_WORD-1$                    ; code for word dtype
                 03EF'     29F8    785         .WORD   FLOAT_TO_LONG-1$                    ; code for long dtype
                 D6C6      29FA    786         .WORD   ERR_DATTYPERR-1$                    ; quad not supported
                 05D0'     29FC    787         .WORD   FLOAT_TO_FLOAT-1$                   ; code for float dtype
                 07AE'     29FE    788         .WORD   FLOAT_TO_DOUBL-1$                   ; code for double dtype
                           2A00    789 ;+
                           2A00    790 ; Check for g and h floating separately, since their dtypes do not fit into
                           2A00    791 ; the CASEB range.
                           2A00    792 ;-
      1B    02 A8    91   2A00    793         CMPB    DSC$B_DTYPE(R8), #DSC$K_DTYPE_G
               03    12   2A04    794         BNEQ    2$
             0990    31   2A06    795         BRW     FLOAT_TO_GFLOA                      ; code for gfloat dtype
                          2A09    796
      1C    02 A8    91   2A09    797 2$:     CMPB    DSC$B_DTYPE(R8), #DSC$K_DTYPE_H
               03    12   2A0D    798         BNEQ    3$
             0B6E    31   2A0F    799         BRW     FLOAT_TO_HFLOA                      ; code for hfloat dtype
                          2A12    800
      18    02 A8    91   2A12    801 3$:     CMPB    DSC$B_DTYPE(R8), #DSC$K_DTYPE_DSC
               06    12   2A16    802         BNEQ    4$
      58    04 A8    D0   2A18    803         MOVL    4(R8), R8                           ; get addr of desc
               D1    11   2A1C    804         BRB     5$                                  ; CASE again for dtype in desc
                          2A1E    805
             D699    31   2A1E    806 4$:     BRW     ERR_DATTYPERR
                          2A21    807
                          2A21    808 ;+
                          2A21    809 ; Now type of source and destination arrays are known.  Use the macro to
                          2A21    810 ; generate the code for each case
                          2A21    811 ;-
                          2A21    812
```

```
                2A21    814 FLOAT_TO_BYTE:  $BAS$MAT_ASS    F, B
                2C02    815
```

```
                2C02    817 FLOAT_TO_WORD:  $BAS$MAT_ASS    F, W
                2DE3    818
```

```
2DE3    820 FLOAT_TO_LONG:  $BAS$MAT_ASS    F, L
2FC4    821
```

```
                823 FLOAT_TO_FLOAT: $BAS$MAT_ASS    F, F
                824
```

```
                826 FLOAT_TO_DOUBL: $BAS$MAT_ASS    F, D
                827
```

```
3399    829 FLOAT_TO_GFLOA: $BAS$MAT_ASS    F, G
3580    830
```

```
3580    832 FLOAT_TO_HFLOA: $BAS$MAT_ASS    F, H
3767    833
```

```
                                3767    835 ;+
                                3767    836 ; Source array is a double array.  Now differentiate on the destination type.
                                3767    837 ;-
                                3767    838
      58    08 AC    D0        3767    839 DOUBLE: MOVL    dest_matrix(AP), R8             ; point to dest descriptor
05    06    02 A8    8F        376B    840 5$:     CASEB   DSC$B_DTYPE(R8), #DSC$K_DTYPE_B, #<DSC$K_DTYPE_D - DSC$K_DTYPE_B>
                     002D'     3770    841 1$:     .WORD   DOUBLE_TO_BYTE-1$               ; code for byte dtype
                     021F'     3772    842         .WORD   DOUBLE_TO_WORD-1$               ; code for word dtype
                     0411'     3774    843         .WORD   DOUBLE_TO_LONG-1$               ; code for long dtype
                     C94A      3776    844         .WORD   ERR_DATTYPERR-1$                ; quad not supported
                     0603'     3778    845         .WORD   DOUBLE_TO_FLOA-1$               ; code for float dtype
                     07F5'     377A    846         .WORD   DOUBLE_TO_DOUBL-1$              ; code for double dtype
                               377C    847 ;+
                               377C    848 ; Check for g and h floating separately, since their dtypes do not fit into
                               377C    849 ; the CASEB range.
                               377C    850 ;-
      1B    02 A8    91        377C    851         CMPB    DSC$B_DTYPE(R8), #DSC$K_DTYPE_G
               03    12        3780    852         BNEQ    2$
            09BE     31        3782    853         BRW     DOUBLE_TO_GFLOA                 ; code for gfloat dtype
                               3785    854
      1C    02 A8    91        3785    855 2$:     CMPB    DSC$B_DTYPE(R8), #DSC$K_DTYPE_H
               03    12        3789    856         BNEQ    3$
            0BB1     31        378B    857         BRW     DOUBLE_TO_HFLOA                 ; code for hfloat dtype
                               378E    858
      18    02 A8    91        378E    859 3$:     CMPB    DSC$B_DTYPE(R8), #DSC$K_DTYPE_DSC
               06    12        3792    860         BNEQ    4$
      58    04 A8    D0        3794    861         MOVL    4(R8), R8                       ; get addr of desc
               D1    11        3798    862         BRB     5$                              ; CASE again for dtype in desc
                               379A    863
            C91D     31        379A    864 4$:     BRW     ERR_DATTYPERR
                               379D    865
                               379D    866
                               379D    867 ;+
                               379D    868 ; Now type of source and destination arrays are known.  Use the macro to
                               379D    869 ; generate the code for each case
                               379D    870 ;-
                               379D    871
```

```
379D    873 DOUBLE_TO_BYTE: $BAS$MAT_ASS    D, B
398F    874
```

```
                398F    876 DOUBLE_TO_WORD: $BAS$MAT_ASS    D, W
                3881    877
```

```
                3B81    879 DOUBLE_TO_LONG: $BAS$MAT_ASS    D, L
                3D73    880
```

```
                3D73    882 DOUBLE_TO_FLOA: $BAS$MAT_ASS    D, F
                3F65    883
```

```
                        885 DOUBLE_TO_DOUBL: $BAS$MAT_ASS   D, D
                        886
```

```
                888 DOUBLE_TO_GFLOA: $BAS$MAT_ASS   D, G
                889
```

```
                        433F    891 DOUBLE_TO_HFLOA: $BAS$MAT_ASS   D, H
                        4537    892
                        4537    893 ;+
                        4537    894 ; Source array is a gfloat array.  Now differentiate on the destination type.
                        4537    895 ;-
                        4537    896
      58    08 AC   D0  4537    897 GFLOAT: MOVL    dest_matrix(AP), R8                 ; point to dest descriptor
05    06    02 A8   8F  453B    898 5$:     CASEB   DSC$B_DTYPE(R8), #DSC$K_DTYPE_B, #<DSC$K_DTYPE_D - DSC$K_DTYPE_B>
                002D' 4540    899 1$:     .WORD   GFLOAT_TO_BYTE-1$                   ; code for byte dtype
                0215' 4542    900         .WORD   GFLOAT_TO_WORD-1$                   ; code for word dtype
                03FD' 4544    901         .WORD   GFLOAT_TO_LONG-1$                   ; code for long dtype
                BB7A  4546    902         .WORD   ERR_DATTYPERR-1$                    ; quad not supported
                05E5' 4548    903         .WORD   GFLOAT_TO_FLOAT-1$                  ; code for float dtype
                07CD' 454A    904         .WORD   GFLOAT_TO_DOUBL-1$                  ; code for double dtype
                        454C    905
                        454C    906 ;+
                        454C    907 ; Check for g and h floating separately, since their dtypes do not fit into
                        454C    908 ; the CASEB range.
                        454C    909 ;-
      1B    02 A8   91  454C    910         CMPB    DSC$B_DTYPE(R8), #DSC$K_DTYPE_G
               03   12  4550    911         BNEQ    2$
             09A4   31  4552    912         BRW     GFLOAT_TO_GFLOA                     ; code for gfloat dtype
                        4555    913
      1C    02 A8   91  4555    914 2$:     CMPB    DSC$B_DTYPE(R8), #DSC$K_DTYPE_H
               03   12  4559    915         BNEQ    3$
             0B84   31  455B    916         BRW     GFLOAT_TO_HFLOA                     ; code for hfloat dtype
                        455E    917
      18    02 A8   91  455E    918 3$:     CMPB    DSC$B_DTYPE(R8), #DSC$K_DTYPE_DSC
               06   12  4562    919         BNEQ    4$
      58    04 A8   D0  4564    920         MOVL    4(R8), R8                           ; get addr of desc
               D1   11  4568    921         BRB     5$                                  ; CASE again for dtype in desc
                        456A    922
             BB4D   31  456A    923 4$:     BRW     ERR_DATTYPERR
                        456D    924
                        456D    925 ;+
                        456D    926 ; Now type of source and destination arrays are known.  Use the macro to
                        456D    927 ; generate the code for each case
                        456D    928 ;-
                        456D    929
```

```
                    931 GFLOAT_TO_BYTE: $BAS$MAT_ASS    G, B
                    932
```

```
                4755    934 GFLOAT_TO_WORD: $BAS$MAT_ASS    G, W
                493D    935
```

```
                493D    937 GFLOAT_TO_LONG: $BAS$MAT_ASS    G, L
                4B25    938
```

```
                4B25    940 GFLOAT_TO_FLOAT:$BAS$MAT_ASS    G, F
                4D0D    941
```

```
4D0D   943 GFLOAT_TO_DOUBL:$BAS$MAT_ASS     G, D
4EF9   944
```

```
                4EF9    946 GFLOAT_TO_GFLOA:$BAS$MAT_ASS    G, G
                50E2    947
```

```
                        50E2    949 GFLOAT_TO_HFLOA:$BAS$MAT_ASS    G, H
                        52CF    950 ;+
                        52CF    951 ; Source array is an hfloat array.  Now differentiate on the destination type.
                        52CF    952 ;-
                        52CF    953
        58    08 AC   D0  52CF    954 HFLOAT: MOVL    dest_matrix(AP), R8                 ; point to dest descriptor
05      06    02 A8   8F  52D3    955 5$:     CASEB   DSC$B_DTYPE(R8), #DSC$K_DTYPE_B, #<DSC$K_DTYPE_D - DSC$K_DTYPE_B>
                002D' 52D8    956 1$:     .WORD   HFLOAT_TO_BYTE-1$                   ; code for byte dtype
                0215' 52DA    957         .WORD   HFLOAT_TO_WORD-1$                   ; code for word dtype
                03FD' 52DC    958         .WORD   HFLOAT_TO_LONG-1$                   ; code for long dtype
                ADE2  52DE    959         .WORD   ERR_DATTYPERR-1$                    ; quad not supported
                05E5' 52E0    960         .WORD   HFLOAT_TO_FLOA-1$                   ; code for float dtype
                07CD' 52E2    961         .WORD   HFLOAT_TO_DOUBL-1$                  ; code for double dtype
                        52E4    962
                        52E4    963 ;+
                        52E4    964 ; Check for g and h floating separately, since their dtypes do not fit into
                        52E4    965 ; the CASEB range.
                        52E4    966 ;-
        1B    02 A8   91  52E4    967         CMPB    DSC$B_DTYPE(R8), #DSC$K_DTYPE_G
                 03   12  52E8    968         BNEQ    2$
                09B6  31  52EA    969         BRW     HFLOAT_TO_GFLOA                 ; code for gfloat dtype
                        52ED    970
        1C    02 A8   91  52ED    971 2$:     CMPB    DSC$B_DTYPE(R8), #DSC$K_DTYPE_H
                 03   12  52F1    972         BNEQ    3$
                0B9A  31  52F3    973         BRW     HFLOAT_TO_HFLOA                 ; code for hfloat dtype
                        52F6    974
        18    02 A8   91  52F6    975 3$:     CMPB    DSC$B_DTYPE(R8), #DSC$K_DTYPE_DSC
                 06   12  52FA    976         BNEQ    4$
        58    04 A8   D0  52FC    977         MOVL    4(R8), R8                       ; get addr of desc
                 D1   11  5300    978         BRB     5$                              ; CASE again for dtype in desc
                        5302    979
                ADB5  31  5302    980 4$:     BRW     ERR_DATTYPERR
                        5305    981
                        5305    982 ;+
                        5305    983 ; Now type of source and destination arrays are known.  Use the macro to
                        5305    984 ; generate the code for each case
                        5305    985 ;-
                        5305    986
```

```
                5305    988 HFLOAT_TO_BYTE: $BAS$MAT_ASS    H, B
                54ED    989
```

```
54ED    991 HFLOAT_TO_WORD: $BAS$MAT_ASS    H, W
56D5    992
```

```
                56D5    994 HFLOAT_TO_LONG: $BAS$MAT_ASS    H, L
                58BD    995
```

```
588D    997 HFLOAT_TO_FLOA: $BAS$MAT_ASS    H, F
5AA5    998
```

```
                5AA5  1000 HFLOAT_TO_DOUBL:$BAS$MAT_ASS     H, D
                5CA3  1001
```

```
                5CA3  1003 HFLOAT_TO_GFLOA:$BAS$MAT_ASS    H, G
                5E90  1004
```

```
                5E90  1006 HFLOAT_TO_HFLOA:$BAS$MAT_ASS    H, H
                6079  1007          .END                                ; end of BAS$MAT_ASSIGN
```

```
BAS$$SCALE_R1                   ********    X   00
BAS$$STOP                       ********    X   00
BAS$FETCH_BFA                   ********    X   00
BAS$FET_FA_B_R8                 ********    X   00
BAS$FET_FA_D_R8                 ********    X   00
BAS$FET_FA_F_R8                 ********    X   00
BAS$FET_FA_G_R8                 ********    X   00
BAS$FET_FA_H_R8                 ********    X   00
BAS$FET_FA_L_R8                 ********    X   00
BAS$FET_FA_W_R8                 ********    X   00
BAS$K_ARGDONMAT                 ********    X   00
BAS$K_DATTYPERR                 ********    X   00
BAS$MAT_ASSIGN                  00000000 RG     02
BAS$MAT_REDIM                   ********    X   00
BAS$STORE_BFA                   ********    X   00
BAS$STO_FA_B_R8                 ********    X   00
BAS$STO_FA_D_R8                 ********    X   00
BAS$STO_FA_F_R8                 ********    X   00
BAS$STO_FA_G_R8                 ********    X   00
BAS$STO_FA_H_R8                 ********    X   00
BAS$STO_FA_L_R8                 ********    X   00
BAS$STO_FA_W_R8                 ********    X   00
BYTE                            00000177 R      02
BYTE_TO_BYTE                    000001AD R      02
BYTE_TO_DOUBLE                  0000092E R      02
BYTE_TO_FLOAT                   0000074D R      02
BYTE_TO_GFLOAT                  00000B25 R      02
BYTE_TO_HFLOAT                  00000D0C R      02
BYTE_TO_LONG                    0000056C R      02
BYTE_TO_WORD                    0000038B R      02
CLASS                       =   0000000F
DATA                        =   00000014
DEST_MATRIX                 =   00000008
DOUBLE                          00003767 R      02
DOUBLE_TO_BYTE                  0000379D R      02
DOUBLE_TO_DOUBL                 00003F65 R      02
DOUBLE_TO_FLOA                  00003D73 R      02
DOUBLE_TO_GFLOA                 00004143 R      02
DOUBLE_TO_HFLOA                 0000433F R      02
DOUBLE_TO_LONG                  00003B81 R      02
DOUBLE_TO_WORD                  0000398F R      02
DSC$A_A0                    =   00000010
DSC$B_AFLAGS                =   0000000A
DSC$B_CLASS                 =   00000003
DSC$B_DIMCT                 =   0000000B
DSC$B_DTYPE                 =   00000002
DSC$K_CLASS_BFA             =   000000BF
DSC$K_CLASS_D               =   00000002
DSC$K_DTYPE_B               =   00000006
DSC$K_DTYPE_D               =   0000000B
DSC$K_DTYPE_DSC             =   00000018
DSC$K_DTYPE_G               =   0000001B
DSC$K_DTYPE_H               =   0000001C
DSC$K_DTYPE_T               =   0000000E
DSC$L_L1_1                  =   00000018
DSC$L_L1_2                  =   0000001C
DSC$L_L2_2                  =   00000024
DSC$L_M1                    =   00000014
DSC$L_M2                    =   00000018
DSC$L_U1_1                  =   0000001C
DSC$L_U1_2                  =   00000020
DSC$L_U2_2                  =   00000028
DSC$V_FL_BOUNDS             =   00000007
DSC$W_LENGTH                =   00000000
DTYPE                       =   0000000E
ERR_ARGDONMAT                   00000035 R      02
ERR_DATTYPERR                   000000BA R      02
FIRST_ARG                   =   00000004
FLOAT                           000029EB R      02
FLOAT_TO_BYTE                   00002A21 R      02
FLOAT_TO_DOUBL                  000031A2 R      02
FLOAT_TO_FLOAT                  00002FC4 R      02
FLOAT_TO_GFLOA                  00003399 R      02
FLOAT_TO_HFLOA                  00003580 R      02
FLOAT_TO_LONG                   00002DE3 R      02
FLOAT_TO_WORD                   00002C02 R      02
GFLOAT                          00004537 R      02
GFLOAT_TO_BYTE                  0000456D R      02
GFLOAT_TO_DOUBL                 00004D0D R      02
GFLOAT_TO_FLOAT                 00004B25 R      02
GFLOAT_TO_GFLOA                 00004EF9 R      02
GFLOAT_TO_HFLOA                 000050E2 R      02
GFLOAT_TO_LONG                  0000493D R      02
GFLOAT_TO_WORD                  00004755 R      02
HFLOAT                          000052CF R      02
HFLOAT_TO_BYTE                  00005305 R      02
HFLOAT_TO_DOUBL                 00005AA5 R      02
HFLOAT_TO_FLOA                  000058BD R      02
HFLOAT_TO_GFLOA                 00005CA3 R      02
HFLOAT_TO_HFLOA                 00005E90 R      02
HFLOAT_TO_LONG                  000056D5 R      02
HFLOAT_TO_WORD                  000054ED R      02
INDEX1                      =   0000000C
INDEX2                      =   00000010
INIT_ONE_SUB                    00000042 R      02
INIT_ONE_SUB_S                  000000E5 R      02
INIT_TWO_SUBS                   00000061 R      02
INIT_TWO_SUBS_S                 00000108 R      02
LONG                            00001C6F R      02
LONG_TO_BYTE                    00001CA5 R      02
LONG_TO_DOUBLE                  00002426 R      02
LONG_TO_FLOAT                   00002245 R      02
LONG_TO_GFLOAT                  0000261D R      02
LONG_TO_HFLOAT                  00002804 R      02
LONG_TO_LONG                    00002067 R      02
LONG_TO_WORD                    00001E86 R      02
LOOP_1ST_SUBBB                  000001AD R      02
LOOP_1ST_SUBBD                  0000092E R      02
LOOP_1ST_SUBBF                  0000074D R      02
LOOP_1ST_SUBBG                  00000B25 R      02
LOOP_1ST_SUBBH                  00000D0C R      02
LOOP_1ST_SUBBL                  0000056C R      02
LOOP_1ST_SUBBW                  0000038B R      02
LOOP_1ST_SUBDB                  0000379D R      02
```

```
LOOP_1ST_SUBDD  00003F65 R  02
LOOP_1ST_SUBDF  00003D73 R  02
LOOP_1ST_SUBDG  00004143 R  02
LOOP_1ST_SUBDH  0000433F R  02
LOOP_1ST_SUBDL  00003B81 R  02
LOOP_1ST_SUBDW  0000398F R  02
LOOP_1ST_SUBFB  00002A21 R  02
LOOP_1ST_SUBFD  000031A2 R  02
LOOP_1ST_SUBFF  00002FC4 R  02
LOOP_1ST_SUBFG  00003399 R  02
LOOP_1ST_SUBFH  00003580 R  02
LOOP_1ST_SUBFL  00002DE3 R  02
LOOP_1ST_SUBFW  00002C02 R  02
LOOP_1ST_SUBGB  0000456D R  02
LOOP_1ST_SUBGD  00004D0D R  02
LOOP_1ST_SUBGF  00004B25 R  02
LOOP_1ST_SUBGG  00004EF9 R  02
LOOP_1ST_SUBGH  000050E2 R  02
LOOP_1ST_SUBGL  0000493D R  02
LOOP_1ST_SUBGW  00004755 R  02
LOOP_1ST_SUBHB  00005305 R  02
LOOP_1ST_SUBHD  00005AA5 R  02
LOOP_1ST_SUBHF  000058BD R  02
LOOP_1ST_SUBHG  00005CA3 R  02
LOOP_1ST_SUBHH  00005E90 R  02
LOOP_1ST_SUBHL  000056D5 R  02
LOOP_1ST_SUBHW  000054ED R  02
LOOP_1ST_SUBLB  00001CA5 R  02
LOOP_1ST_SUBLD  00002426 R  02
LOOP_1ST_SUBLF  00002245 R  02
LOOP_1ST_SUBLG  0000261D R  02
LOOP_1ST_SUBLH  00002804 R  02
LOOP_1ST_SUBLL  00002067 R  02
LOOP_1ST_SUBLW  00001E86 R  02
LOOP_1ST_SUBWB  00000F29 R  02
LOOP_1ST_SUBWD  000016AA R  02
LOOP_1ST_SUBWF  000014C9 R  02
LOOP_1ST_SUBWG  000018A1 R  02
LOOP_1ST_SUBWH  00001A88 R  02
LOOP_1ST_SUBWL  000012E8 R  02
LOOP_1ST_SUBWW  0000110A R  02
LOOP_1ST_SUB_S  00000133 R  02
LOOP_2ND_SUBBB  00001B0 R  02
LOOP_2ND_SUBBD  00000931 R  02
LOOP_2ND_SUBBF  00000750 R  02
LOOP_2ND_SUBBG  00000B28 R  02
LOOP_2ND_SUBBH  00000D0F R  02
LOOP_2ND_SUBBL  0000056F R  02
LOOP_2ND_SUBBW  0000038E R  02
LOOP_2ND_SUBDB  000037A0 R  02
LOOP_2ND_SUBDD  00003F68 R  02
LOOP_2ND_SUBDF  00003D76 R  02
LOOP_2ND_SUBDG  00004146 R  02
LOOP_2ND_SUBDH  00004342 R  02
LOOP_2ND_SUBDL  00003B84 R  02
LOOP_2ND_SUBDW  00003992 R  02
LOOP_2ND_SUBFB  00002A24 R  02
LOOP_2ND_SUBFD  000031A5 R  02
LOOP_2ND_SUBFF  00002FC7 R  02
LOOP_2ND_SUBFG  0000339C R  02
LOOP_2ND_SUBFH  00003583 R  02
LOOP_2ND_SUBFL  00002DE6 R  02
LOOP_2ND_SUBFW  00002C05 R  02
LOOP_2ND_SUBGB  00004570 R  02
LOOP_2ND_SUBGD  00004D10 R  02
LOOP_2ND_SUBGF  00004B28 R  02
LOOP_2ND_SUBGG  00004EFC R  02
LOOP_2ND_SUBGH  000050E5 R  02
LOOP_2ND_SUBGL  00004940 R  02
LOOP_2ND_SUBGW  00004758 R  02
LOOP_2ND_SUBHB  00005308 R  02
LOOP_2ND_SUBHD  00005AA8 R  02
LOOP_2ND_SUBHF  000058C0 R  02
LOOP_2ND_SUBHG  00005CA6 R  02
LOOP_2ND_SUBHH  00005E93 R  02
LOOP_2ND_SUBHL  000056D8 R  02
LOOP_2ND_SUBHW  000054F0 R  02
LOOP_2ND_SUBLB  00001CA8 R  02
LOOP_2ND_SUBLD  00002429 R  02
LOOP_2ND_SUBLF  00002248 R  02
LOOP_2ND_SUBLG  00002620 R  02
LOOP_2ND_SUBLH  00002807 R  02
LOOP_2ND_SUBLL  0000206A R  02
LOOP_2ND_SUBLW  00001E89 R  02
LOOP_2ND_SUBWB  00000F2C R  02
LOOP_2ND_SUBWD  000016AD R  02
LOOP_2ND_SUBWF  000014CC R  02
LOOP_2ND_SUBWG  000018A4 R  02
LOOP_2ND_SUBWH  00001A8B R  02
LOOP_2ND_SUBWL  000012EB R  02
LOOP_2ND_SUBWW  0000110D R  02
LOOP_2ND_SUB_S  00000136 R  02
LOWER_BND1  = 00000004
LOWER_BND2  = 00000000
MTH$DINT_R4  ******** X  00
POINTER  = 00000010
SECOND_ARG  = 00000008
SEPARATE_DTYPES  00000088 R  02
SF$L_SAVE_FP  = 0000000C
SRC_MATRIX  = 00000004
STR$FREE1_DX_R4  ******** X  00
STRING  000000C7 R  02
STR_LEN  = 0000000C
TEMP_DESC  = 00000014
UPPER_BND1  = 00000008
VALUE_DESC  = 0000000C
WORD  00000EF3 R  02
WORD_TO_BYTE  00000F29 R  02
WORD_TO_DOUBL  000016AA R  02
WORD_TO_FLOAT  000014C9 R  02
WORD_TO_GFLOAT  000018A1 R  02
WORD_TO_HFLOAT  00001A88 R  02
WORD_TO_LONG  000012E8 R  02
WORD_TO_WORD  0000110A 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	00006079 (24697.)	02 (2.)	PIC USR CON REL LCL SHR EXE RD NOWRT NOVEC LONG

Performance indicators

Phase	Page faults	CPU Time	Elapsed Time
Initialization	31	00:00:00.09	00:00:01.05
Command processing	117	00:00:00.60	00:00:02.77
Pass 1	804	00:00:41.03	00:01:23.33
Symbol table sort	0	00:00:01.80	00:00:02.04
Pass 2	328	00:00:09.76	00:00:25.12
Symbol table output	27	00:00:00.18	00:00:00.40
Psect synopsis output	3	00:00:00.02	00:00:00.02
Cross-reference output	0	00:00:00.00	00:00:00.00
Assembler run totals	1312	00:00:53.49	00:01:54.74

The working set limit was 2000 pages.
301438 bytes (589 pages) of virtual memory were used to buffer the intermediate code.
There were 60 pages of symbol table space allocated to hold 370 non-local and 829 local symbols.
1007 source lines were read in Pass 1, producing 76 object records in Pass 2.
32 pages of virtual memory were used to define 11 macros.

Macro library statistics

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

493 GETS were required to define 7 macros.

There were no errors, warnings or information messages.

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

BASMATINT
LIS

BASMATION
LIS

BASMATASS
LIS