```
MMM          MMM  TTTTTTTTTTTTTTT   HHH          HHH  RRRRRRRRRRR     TTTTTTTTTTTTTTT   LLL
MMM          MMM  TTTTTTTTTTTTTTT   HHH          HHH  RRRRRRRRRRRR    TTTTTTTTTTTTTTT   LLL
MMM          MMM  TTTTTTTTTTTTTTT   HHH          HHH  RRRRRRRRRRRR    TTTTTTTTTTTTTTT   LLL
MMMMMM    MMMMMM        TTT         HHH          HHH  RRR         RRR       TTT         LLL
MMMMMM    MMMMMM        TTT         HHH          HHH  RRR         RRR       TTT         LLL
MMMMMM    MMMMMM        TTT         HHH          HHH  RRR         RRR       TTT         LLL
MMM   MMM   MMM         TTT         HHH          HHH  RRR         RRR       TTT         LLL
MMM   MMM   MMM         TTT         HHH          HHH  RRR         RRR       TTT         LLL
MMM   MMM   MMM         TTT         HHH          HHH  RRR         RRR       TTT         LLL
MMM         MMM         TTT         HHHHHHHHHHHHHHHH  RRRRRRRRRRR           TTT         LLL
MMM         MMM         TTT         HHHHHHHHHHHHHHHH  RRRRRRRRRRRR          TTT         LLL
MMM         MMM         TTT         HHHHHHHHHHHHHHHH  RRRRRRRRRRRR          TTT         LLL
MMM         MMM         TTT         HHH          HHH  RRR   RRR             TTT         LLL
MMM         MMM         TTT         HHH          HHH  RRR   RRR             TTT         LLL
MMM         MMM         TTT         HHH          HHH  RRR   RRR             TTT         LLL
MMM         MMM         TTT         HHH          HHH  RRR      RRR          TTT         LLL
MMM         MMM         TTT         HHH          HHH  RRR      RRR          TTT         LLL
MMM         MMM         TTT         HHH          HHH  RRR      RRR          TTT         LLL
MMM         MMM         TTT         HHH          HHH  RRR         RRR       TTT         LLLLLLLLLLLLLLLL
MMM         MMM         TTT         HHH          HHH  RRR         RRR       TTT         LLLLLLLLLLLLLLLL
MMM         MMM         TTT         HHH          HHH  RRR         RRR       TTT         LLLLLLLLLLLLLLLL
```

```
MM       MM  TTTTTTTTTT  HH       HH  DDDDDDDD    MM       MM    OOOOOO    DDDDDDDD
MM       MM  TTTTTTTTTT  HH       HH  DDDDDDDD    MM       MM    OOOOOO    DDDDDDDD
MMMM   MMMM      TT      HH       HH  DD       DD  MMMM   MMMM  OO      OO  DD       DD
MMMM   MMMM      TT      HH       HH  DD       DD  MMMM   MMMM  OO      OO  DD       DD
MM  MM  MM       TT      HH       HH  DD       DD  MM  MM  MM  OO      OO  DD       DD
MM  MM  MM       TT      HH       HH  DD       DD  MM  MM  MM  OO      OO  DD       DD
MM      MM       TT      HHHHHHHHHHH  DD       DD  MM      MM  OO      OO  DD       DD
MM      MM       TT      HHHHHHHHHHH  DD       DD  MM      MM  OO      OO  DD       DD
MM      MM       TT      HH       HH  DD       DD  MM      MM  OO      OO  DD       DD
MM      MM       TT      HH       HH  DD       DD  MM      MM  OO      OO  DD       DD
MM      MM       TT      HH       HH  DD       DD  MM      MM  OO      OO  DD       DD      ....
MM      MM       TT      HH       HH  DD       DD  MM      MM  OO      OO  DD       DD      ....
MM      MM       TT      HH       HH  DDDDDDDD    MM      MM    OOOOOO    DDDDDDDD        ....
MM      MM       TT      HH       HH  DDDDDDDD    MM      MM    OOOOOO    DDDDDDDD        ....

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
```

```
0000     1          .TITLE  MTH$DMOD
0000     2          .IDENT  /3-001/                 ; File: MTHDMOD.MAR Edit: JCW3001
0000     3
0000     4 ;
0000     5 ;*****************************************************************************
0000     6 ;*                                                                           *
0000     7 ;*  COPYRIGHT (c) 1978, 1980, 1982, 1984 BY                                  *
0000     8 ;*  DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.                   *
0000     9 ;*  ALL RIGHTS RESERVED.                                                     *
0000    10 ;*                                                                           *
0000    11 ;*  THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED    *
0000    12 ;*  ONLY IN  ACCORDANCE WITH  THE  TERMS  OF  SUCH  LICENSE  AND WITH THE   *
0000    13 ;*  INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR  ANY  OTHER    *
0000    14 ;*  COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY    *
0000    15 ;*  OTHER PERSON.  NO TITLE TO AND OWNERSHIP OF  THE  SOFTWARE IS  HEREBY    *
0000    16 ;*  TRANSFERRED.                                                             *
0000    17 ;*                                                                           *
0000    18 ;*  THE INFORMATION IN THIS SOFTWARE IS  SUBJECT TO CHANGE WITHOUT NOTICE    *
0000    19 ;*  AND  SHOULD  NOT  BE  CONSTRUED AS  A COMMITMENT BY DIGITAL EQUIPMENT   *
0000    20 ;*  CORPORATION.                                                             *
0000    21 ;*                                                                           *
0000    22 ;*  DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE  OR  RELIABILITY OF ITS    *
0000    23 ;*  SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.                  *
0000    24 ;*                                                                           *
0000    25 ;*                                                                           *
0000    26 ;*****************************************************************************
0000    27 ;
0000    28
0000    29 ;++
0000    30 ; FACILITY: MATH LIBRARY
0000    31 ;
0000    32 ; ABSTRACT:
0000    33 ;
0000    34 ;       This module contains the routine MTH$DMOD:
0000    35 ;       It returns the remainder of the division of arg1/arg2 using
0000    36 ;       the following equation:
0000    37 ;               arg1 - (int(arg1/arg2))*arg2
0000    38 ;
0000    39 ;
0000    40 ;--
0000    41 ;
0000    42 ; AUTHOR: Jeffrey C. Wiener, CREATION DATE: 21-DEC-1982
0000    43 ;
0000    44 ; MODIFIED BY:
0000    45 ;
0000    46 ;
0000    47 ;--
0000    48
0000    49          .SBTTL  HISTORY                 ; Detailed Current Edit History
0000    50 ;
0000    51 ; 3-001 Original version of complete re-write          JCW  21-DEC-82
0000    52 ;
```

```
                     0000     54          .SBTTL  DECLARATIONS
                     0000     55 ;
                     0000     56 ; INCLUDE FILES:
                     0000     57 ;
                     0000     58 ;       NONE
                     0000     59 ;
                     0000     60 ; EXTERNAL SYMBOLS:
                     0000     61 ;
                     0000     62          .DSABL  GBL             ; Force all external symbols to be declared
                     0000     63          .EXTRN  MTH$$SIGNAL
                     0000     64          .EXTRN  MTH$K_FLOUNDMAT
                     0000     65          .EXTRN  MTH$K_INVARGMAT
                     0000     66 ;
                     0000     67 ; LIBRARY MACROS CALLS:
                     0000     68 ;
                     0000     69          $SFDEF                          ; Define SF$ (stack frame) symbols
                     0000     70 ;
                     0000     71 ; EQUATED SYMBOLS:
                     0000     72 ;
00001B80             0000     73          EXP_55    = ^X00001B80          ; 55*2^7
FFFF0FFF             0000     74          HIGH_MASK = ^XFFFF0FFF
                     0000     75 ;
                     0000     76 ; OWN STORAGE:
                     0000     77 ;
                     0000     78 ;       NONE
                     0000     79 ;
                     0000     80 ; PSECT DECLARATIONS:
                     0000     81 ;
                 00000000     82          .PSECT  _MTH$CODE               PIC, SHR, LONG, EXE, NOWRT
                     0000     83
                     0000     84 ; CONSTANTS:
                     0000     85
                     0000     86 TWO_EXP_55:
00000000 00005C00    0000     87          .LONG   ^X00005C00, ^X0         ; 2**55
```

```
                     0008    89          .SBTTL  MTH$DMOD - D REAL*8 remainder
                     0008    90 ;++
                     0008    91 ; FUNCTIONAL DESCRIPTION:
                     0008    92 ;
                     0008    93 ;       Return the remainder of arg1/arg2 in D_floating point format
                     0008    94 ;       Remainder = arg1 - (int(arg1/arg2))*arg2
                     0008    95 ;
                     0008    96 ; The algorithm used to evaluate the DMOD function is as follows:
                     0008    97 ;
                     0008    98 ;               X = the first argument.
                     0008    99 ;               Y = the second argument.
                     0008   100 ;       step 1. m = the exponent of Y.
                     0008   101 ;               n = the exponent of X.
                     0008   102 ;               c = n - m
                     0008   103 ;               If c < 0, end with result = X.
                     0008   104 ;       step 2. I = the fractional part of X.
                     0008   105 ;               J = the fractional part of Y.
                     0008   106 ;               If I >= J, I = I - J
                     0008   107 ;               Go to step 5.
                     0008   108 ;       step 3. L = 2^(p-1)*I, where p = 56 for D_floating numbers.
                     0008   109 ;       step 4. T = L/J
                     0008   110 ;               T = [T+2^(p-1)]-2^(p-1).   T is int(L/J) or int(L/J)+1
                     0008   111 ;               I = L - J * T
                     0008   112 ;               If I < 0, I = I + J        T was int(L/J)+1
                     0008   113 ;       step 5. c = c - (p-1)
                     0008   114 ;               If c > 0 go to step 3.
                     0008   115 ;       step 6. If c = -(p-1) go to step 9.
                     0008   116 ;       step 7. L = 2^(p-1+c) * I
                     0008   117 ;       step 8. I = L - J * T
                     0008   118 ;       step 9. Result = 2^m * I
                     0008   119 ;
                     0008   120 ;
                     0008   121 ; CALLING SEQUENCE:
                     0008   122 ;
                     0008   123 ;       Remainder.wd.v = MTH$DMOD (dividend.rd.r, divisor.rd.r)
                     0008   124 ;
                     0008   125 ; INPUT PARAMETERS:
                     0008   126 ;
                     0008   127 ;       The two input parameters are double precision floating-point
                     0008   128 ;       values. They are passed by reference.
                     0008   129 ;
                     0008   130
00000004             0008   131          DIVIDEND = 4                            ; Dividend = X in the algorithm.
00000008             0008   132          DIVISOR  = 8                            ; Divisor  = Y in the algorithm.
                     0008   133
                     0008   134
                     0008   135 ; IMPLICIT INPUTS:
                     0008   136 ;
                     0008   137 ;       NONE
                     0008   138 ;
                     0008   139 ; FUNCTION VALUE:
                     0008   140 ;
                     0008   141 ;       Remainder of the division of arg1/arg2 is returned as a
                     0008   142 ;       double precision floating point value.
                     0008   143 ;
                     0008   144 ; IMPLICIT OUTPUTS:
                     0008   145 ;
```

```
                                0008   146 ;       NONE
                                0008   147 ;
                                0008   148 ; COMPLETION CODES:
                                0008   149 ;
                                0008   150 ;       NONE
                                0008   151 ;
                                0008   152 ; SIDE EFFECTS:
                                0008   153 ;
                                0008   154 ;       MTH$_INVARGMAT - Invalid argument to math library if the divisor is zero.
                                0008   155 ;       MTH$_FLOUNDMAT - Floating underflow in math library is signaled if
                                0008   156 ;          the FU bit is set in the callers PSL.
                                0008   157 ;
                                0008   158 ;--
                                0008   159
                         01FC   0008   160         .ENTRY  MTH$DMOD,       ^M<R2, R3, R4, R5, R6, R7, R8>
                                000A   161
            52    08 BC    70  000A   162         MOVD    @DIVISOR(AP), R2                ; R2/R3 = Y
                  52       13  000E   163         BEQL    ERROR                           ; Y=0. Division by zero
            50    04 BC    70  0010   164         MOVD    @DIVIDEND(AP), R0               ; R0/R1 = X
                                0014   165
     56     52    FFFF807F 8F  CB  0014   166         BICL3   #^XFFFF807F, R2, R6             ; R6=m is the biased exponent of Y
     58     50    FFFF807F 8F  CB  001C   167         BICL3   #^XFFFF807F, R0, R8             ; R8=n is the biased exponent of X
                                0024   168
               58    56       C2  0024   169         SUBL2   R6, R8                          ; R4 = c = n-m unbiased
                     01       18  0027   170         BGEQ    STEP_2                          ; Result is X if X<Y, ie, if c<0
                              04  0029   171         RET                                     ; R0/R1 = X
                                002A   172
                              56  DD  002A   173 STEP_2: PUSHL   R6                          ; push m onto the stack
                                002C   174
            52    FF80 8F      AA  002C   175         BICW2   #^XFF80, R2                     ; R2/R3 = J = unbiased !fract(Y)!
            52    4000 8F      AC  0031   176         XORW    #^X4000, R2                     ; J = properly biased !fract(Y)!
                                0036   177
            50    FF80 8F      AA  0036   178         BICW2   #^XFF80, R0                     ; R0/R1 = I = unbiased !fract(X)!
            50    4000 8F      AC  003B   179         XORW    #^X4000, R0                     ; I = properly biased !fract(X)!
                                0040   180
                                0040   181 ;+
                                0040   182 ;
                                0040   183 ;       In STEP_4 and STEP_8 the calculation of  I = L   J*int(L/J) must be
                                0040   184 ;       computed as precisely as possible. To do this we will need to write J as
                                0040   185 ;                  J = J1 + J2
                                0040   186 ;       where J1 = the high 24 bits of J and J2 = J - J1, the low 24 bits of J.
                                0040   187 ;
                                0040   188 ;       HIGH_MASK is used to extract the 8 bits of J from longword2 that belong
                                0040   189 ;       to J1.
                                0040   190 ;
                                0040   191 ;-
               7E    52       7D  0040   192         MOVQ    R2, -(SP)                       ; (SP) = J
04 AE     FFFF0FFF 8F          CA  0043   193         BICL    #HIGH_MASK, 4(SP)               ; (SP) = J1 replaces the value
                                004B   194 ;                                                 ;   of J on the top of the SP
            7E    52       6E  63  004B   195         SUBD3   (SP), R2, -(SP)                 ; (SP) = J2 = J - J1
                                004F   196
                  52       50  71  004F   197         CMPD    R0, R2                          ; If I<J
                           63  19  0052   198         BLSS    STEP_5                          ; go to STEP_5
                  50       52  62  0054   199         SUBD2   R2, R0                          ; else I = I-J
                           5E  14  0057   200         BGTR    STEP_5                          ; go to STEP_5 if I>0, or
                                0059   201 ;                                                 ;   else the algorithm ends
                     04 BC    B5  0059   202         TSTW    @DIVIDEND(AP)                   ; the sign of the result is
```

```
                03   18   005C   203          BGEQ    DONE                            ; the same as the sign of
           50   50   72   005E   204          MNEGD   R0, R0                          ; the first argument, A.
                     04   0061   205 DONE:    RET
                          0062   206
      50   01   0F   79   0062   207 ERROR:   ASHQ    #15, #1, R0                     ; Y=0. Reserved operand
        7E   00'8F   9A   0066   208          MOVZBL  #MTH$K_INVARGMAT, -(SP)         ; error code
00000000'GF   01   FB   006A   209          CALLS   #1, G^MTH$$SIGNAL               ; signal the error
                     04   0071   210          RET
                          0072   211
50    00001B80 8F   C0   0072   212 STEP_3:  ADDL2   #EXP_55, R0                    ; R0/R1 = L = 2**(p-1)*I
                          0079   213
                          0079   214 ;+
                          0079   215 ;
                          0079   216 ;       STEP_4:
                          0079   217 ;       2^(p-1) = 2^(55) is added and then subtracted from
                          0079   218 ;       T = int(L/J) to ensure that T = chopped(L/J) or chopped(L/J)+1
                          0079   219 ;
                          0079   220 ;-
                          0079   221
      56   50   52   67   0079   222          DIVD3   R2, R0, R6                      ; R6/R7 = T = L/J
        56   80 AF   60   007D   223          ADDD2   TWO_EXP_55, R6                  ; R6/R7 = T = T+2**(p-1)
      56   FF7B CF   62   0081   224          SUBD2   TWO_EXP_55, R6                  ; T-2**(p-1) = L/J chopped or choppe
                          0086   225
                          0086   226 ;+
                          0086   227 ;
                          0086   228 ;       The calculation of  I = L - J*int(L/J) must be computed as precisely
                          0086   229 ;       as possible. To do this we will need to write T as
                          0086   230 ;                T = Z1 + Z2
                          0086   231 ;       where Z1 = the high 24 bits of T and Z2 = T - Z1, the low 24 bits of T.
                          0086   232 ;
                          0086   233 ;       Now, using J = J1 + J2,
                          0086   234 ;
                          0086   235 ;               L - J * int(L/J) = L - (J1 + J2) * (Z1 + Z2)
                          0086   236 ;                                = L - (Z1 * J1) - (Z1 * J2)
                          0086   237 ;                                    - (Z2 * J1)   (Z2 * J2)
                          0086   238 ;                                = L - (Z1 * J ) = (Z2 * J )
                          0086   239 ;
                          0086   240 ;-
                          0086   241
           54   56   D0   0086   242          MOVL    R6, R4                          ;
55    57   FFFF0FFF 8F   CB   0089   243          BICL3   #HIGH_MASK, R7, R5              ; R4/R5 = Z1
           56   54   62   0091   244          SUBD2   R4, R6                          ; R6/R7 = Z2
      7E   08 AE   54   65   0094   245          MULD3   R4, 8(SP), -(SP)                ; Compute Z1*J1
           50   8E   62   0099   246          SUBD2   (SP)+, R0                       ; R0/R1 = L - Z1*J1
           54   6E   64   009C   247          MULD2   (SP), R4                        ; R4/R5 = Z1*J2
           50   54   62   009F   248          SUBD2   R4, R0                          ; R0/R1 = L - Z1*J
      54   56   08 AE   65   00A2   249          MULD3   8(SP), R6, R4                   ; R4/R5 = Z2*J1
           50   54   62   00A7   250          SUBD2   R4, R0                          ; R0/R1 = L - Z1*J - Z2*J1
           56   6E   64   00AA   251          MULD2   (SP), R6                        ; R6/R7 = Z2*J2
           50   56   62   00AD   252          SUBD2   R6, R0                          ; R0/R1 = L - Z*J
                05   14   00B0   253          BGTR    STEP_5                          ;
                6D   13   00B2   254          BEQL    RETURN                          ; End if R0/R1=0
           50   52   60   00B4   255          ADDD    R2, R0                          ; Add J back in because you had
                          00B7   256                                                  ;   T=chopped(L/J)+1
                          00B7   257
      58   00001B80 8F   C2   00B7   258 STEP_5:  SUBL2   #EXP_55, R8                  ;c = c-(p-1) = c-55
                     B2   18   00BE   259          BGEQ    STEP_3
```

```
                                00C0   260          ADDL2   #EXP_55, R8                     ; c = (p-1)+c = 55+c
                                00C7   261
                                00C7   262 ;+
                                00C7   263 ;
                                00C7   264 ;       The next two lines of code are STEP_6 and STEP_7.
                                00C7   265 ;
                                00C7   266 ;-
                                00C7   267
                          42    00C7   268          BEQL    STEP_9                          ;
                          50    00C9   269          ADDL2   R8, R0                          ; L = I*2^(c+t)
                                00CC   270
                                00CC   271 ;+
                                00CC   272 ;       2^(p-1) = 2^(55) is added and then subtracted from
                                00CC   273 ;       T = int(L/J) to ensure that T = chopped(L/J) or chopped(L/J)+1
                                00CC   274 ;-
                                00CC   275
                    56  50  52  00CC   276          DIVD3   R2, R0, R6                      ; R6/R7 = T = L/J
                    56  FF2C CF 00D0   277          ADDD2   TWO_EXP_55, R6                  ; R6/R7 = T = T+2**(p-1)
                    56  FF27 CF 00D5   278          SUBD2   TWO_EXP_55, R6                  ; T-2**(p-1) = L/J chopped or choppe
                                00DA   279
                                00DA   280 ;+
                                00DA   281 ;
                                00DA   282 ;       STEP_8:
                                00DA   283 ;
                                00DA   284 ;       The calculation of  I = L - J*int(L/J) must be computed as precisel
                                00DA   285 ;       as possible. To do this we will need to write T as
                                00DA   286 ;                T = Z1 + Z2
                                00DA   287 ;       where Z1 = the high 24 bits of T and Z2 = T - Z1, the low 24 bits of T.
                                00DA   288 ;
                                00DA   289 ;       Now, using J = J1 + J2,
                                00DA   290 ;
                                00DA   291 ;                L - J * int(L/J) = L - (J1 + J2) * (Z1 + Z2)
                                00DA   292 ;                                 = L - (Z1 * J1) - (Z1 * J2)
                                00DA   293 ;                                     - (Z2 * J1)   (Z2 * J2)
                                00DA   294 ;                                 = L - (Z1 * J ) = (Z2 * J )
                                00DA   295 ;
                                00DA   296 ;-
                                00DA   297
                    54  56  D0  00DA   298          MOVL    R6, R4                          ;
55  57  FFFF0FFF    8F  CB  00DD   299          BICL3   #HIGH_MASK, R7, R5              ; R4/R5 = Z1
                    56  54  62  00E5   300          SUBD2   R4, R6                          ; R6/R7 = Z2
          7E  08 AE     54  65  00E8   301          MULD3   R4, 8(SP), -(SP)                ; Compute Z1*J1
                    50  8E  62  00ED   302          SUBD2   (SP)+, R0                       ; R0/R1 = L - Z1*J1
                    54  6E  64  00F0   303          MULD2   (SP), R4                        ; R4/R5 = Z1*J2
                    50  54  62  00F3   304          SUBD2   R4, R0                          ; R0/R1 = L - Z1*J
          54  56      08 AE  65  00F6   305          MULD3   8(SP), R6, R4                   ; R4/R5 = Z2*J1
                    50  54  62  00FB   306          SUBD2   R4, R0                          ; R0/R1 = L - Z1*J - Z2*J1
                    56  6E  64  00FE   307          MULD2   (SP), R6                        ; R6/R7 = Z2*J2
                    50  56  62  0101   308          SUBD2   R6, R0                          ; R0/R1 = L - Z*J
                        05  14  0104   309          BGTR    STEP_9                          ;
                        19  13  0106   310          BEQL    RETURN                          ; End if R0/R1=0
                    50  52  60  0108   311          ADDD    R2, R0                          ; Add J back in because you had
                                0108   312                                                  ;   T=chopped(L/J)+1
                                0108   313
          10 AE   4000  8F  A2  010B   314 STEP_9: SUBW    #^X4000, 16(SP)                 ; Remove bias from m and
               50   10 AE  A0  0111   315          ADDW2   16(SP), R0                      ;  form R0/R1 = 2^m*L
                        0B  19  0115   316          BLSS    UNDERFLOW                       ; Branch if underflow
```

```
                        0117   317 TEST_SIGN:
                 04 BC  B5  0117   318          TSTW    @DIVIDEND(AP)                       ; the sign of the result is
                    05  18  011A   319          BGEQ    RETURN                              ; the same as the sign of
           50    8000 8F  A8  011C   320          BISW2   #^X8000, R0                         ; the first argument, X.
                    04  0121   321 RETURN: RET
                        0122   322
                        0122   323 UNDERFLOW:
                    50  7C  0122   324          CLRQ    R0                                  ; Set up default result to 0.0
              0D 04 AD  06  E1  0124   325          BBC     #SF$V_FU, SF$W_SAVE_PSW(FP), NO_FU
                        0129   326                                                          ; Branch if caller has not enabled F
         00000000'8F    DD  0129   327          PUSHL   #MTH$K_FLOUNDMAT                    ; Report MTH$_FLOUNDMAT
     00000000'GF    01  FB  012F   328          CALLS   #1, G^MTH$$SIGNAL                   ; Signal the condition
                    04  0136   329 NO_FU:  RET                                            ; Return
                        0137   330
                        0137   331          .END
```

```
DIVIDEND        =  00000004
DIVISOR         =  00000008
DONE               00000061  R      02
ERROR              00000062  R      02
EXP_55          =  00001B80
HIGH_MASK       =  FFFF0FFF
MTH$$SIGNAL        ********    X    00
MTH$DMOD           00000008  RG     02
MTH$K_FLOUNDMAT    ********     X   00
MTH$K_INVARGMAT    ********     X   00
NO_FU              00000136  R      02
RETURN             00000121  R      02
SF$V_FU         =  00000006
SF$W_SAVE_PSW   =  00000004
STEP_2             0000002A  R      02
STEP_3             00000072  R      02
STEP_5             000000B7  R      02
STEP_9             0000010B  R      02
TEST_SIGN          00000117  R      02
TWO_EXP_55         00000000  R      02
UNDERFLOW          00000122  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
_MTH$CODE	00000137 (311.)	02 (2.)	PIC USR CON REL LCL SHR EXE RD NOWRT NOVEC LONG

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

Phase	Page faults	CPU Time	Elapsed Time
Initialization	34	00:00:00.09	00:00:01.54
Command processing	117	00:00:00.45	00:00:03.28
Pass 1	122	00:00:01.45	00:00:05.82
Symbol table sort	0	00:00:00.03	00:00:00.19
Pass 2	72	00:00:00.71	00:00:03.37
Symbol table output	3	00:00:00.03	00:00:00.07
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	353	00:00:02.79	00:00:14.30

The working set limit was 900 pages.
6542 bytes (13 pages) of virtual memory were used to buffer the intermediate code.
There were 10 pages of symbol table space allocated to hold 48 non-local and 0 local symbols.
331 source lines were read in Pass 1, producing 13 object records in Pass 2.
8 pages of virtual memory were used to define 7 macros.

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

Macro library name                          Macros defined
------------------                          --------------
_$255$DUA28:[SYSLIB]STARLET.MLB;2                    4

88 GETS were required to define 4 macros.

There were no errors, warnings or information messages.

MACRO/ENABLE=SUPPRESSION/DISABLE=(GLOBAL,TRACEBACK)/LIS=LIS$:MTHDMOD/OBJ=OBJ$:MTHDMOD MSRC$:MTHDMOD/UPDATE=(ENH$:MTHDMOD)
```

MTHDCOSH LIS

MTHDMIN LIS

MTHDLOG LIS

MTHDSINCO LIS

MTHDATANH LIS

MTHDNINT LIS

MTHDSQRT LIS

MTHDCONJG LIS

MTHDINT LIS

MTHDMAX1 LIS

MTHDSIGN LIS

MTHDIM LIS

MTHDMOD LIS

MTHDSINH LIS

MTHDEXP LIS

MTHDFLOOR LIS

MTHDPROD LIS