```
MMM         MMM  TTTTTTTTTTTTTTT  HHH         HHH  RRRRRRRRRRRR     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        HHHHHHHHHHHHHHH  RRRRRRRRRRRR           TTT        LLL
MMM         MMM        TTT        HHHHHHHHHHHHHHH  RRRRRRRRRRRR           TTT        LLL
MMM         MMM        TTT        HHHHHHHHHHHHHHH  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        LLLLLLLLLLLLLLL
MMM         MMM        TTT        HHH         HHH  RRR         RRR        TTT        LLLLLLLLLLLLLLL
MMM         MMM        TTT        HHH         HHH  RRR         RRR        TTT        LLLLLLLLLLLLLLL
```

FILEID**OTSPOWDD


```
  OOOOOO    TTTTTTTTTT    SSSSSSSS  PPPPPPPP      OOOOOO   WW        WW  DDDDDDDD    DDDDDDDD
  OOOOOO    TTTTTTTTTT    SSSSSSSS  PPPPPPPP      OOOOOO   WW        WW  DDDDDDDD    DDDDDDDD
OO      OO      TT      SS          PP      PP  OO      OO  WW        WW  DD      DD  DD      DD
OO      OO      TT      SS          PP      PP  OO      OO  WW        WW  DD      DD  DD      DD
OO      OO      TT      SS          PP      PP  OO      OO  WW        WW  DD      DD  DD      DD
OO      OO      TT      SS          PP      PP  OO      OO  WW        WW  DD      DD  DD      DD
OO      OO      TT        SSSSSS    PPPPPPPP    OO      OO  WW        WW  DD      DD  DD      DD
OO      OO      TT        SSSSSS    PPPPPPPP    OO      OO  WW        WW  DD      DD  DD      DD
OO      OO      TT              SS  PP          OO      OO  WW   WW   WW  DD      DD  DD      DD
OO      OO      TT              SS  PP          OO      OO  WW   WW   WW  DD      DD  DD      DD
OO      OO      TT              SS  PP          OO      OO  WWWW  WWWW  DD      DD  DD      DD
OO      OO      TT              SS  PP          OO      OO  WWWW  WWWW  DD      DD  DD      DD    ....
  OOOOOO        TT      SSSSSSSS    PP            OOOOOO    WW        WW  DDDDDDDD    DDDDDDDD      ....
  OOOOOO        TT      SSSSSSSS    PP            OOOOOO    WW        WW  DDDDDDDD    DDDDDDDD      ....
                                                                                                  ....

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

Table of contents

```
0000          .TITLE  OTS$POWDD - DOUBLE PRECISION ** DOUBLE PRECISION power routine
0000          .IDENT  /2-007/          ; File: OTSPOWDD.MAR  Edit: JCW2007
0000
0000  ;
0000  ;****************************************************************************
0000  ;*                                                                          *
0000  ;*  COPYRIGHT (c) 1978, 1980, 1982, 1984 BY                                 *
0000  ;*  DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.                  *
0000  ;*  ALL RIGHTS RESERVED.                                                    *
0000  ;*                                                                          *
0000  ;*  THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED   *
0000  ;*  ONLY IN  ACCORDANCE WITH  THE  TERMS  OF  SUCH  LICENSE  AND WITH THE   *
0000  ;*  INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR  ANY  OTHER   *
0000  ;*  COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY   *
0000  ;*  OTHER PERSON.  NO TITLE TO AND OWNERSHIP OF  THE  SOFTWARE IS  HEREBY   *
0000  ;*  TRANSFERRED.                                                            *
0000  ;*                                                                          *
0000  ;*  THE INFORMATION IN THIS SOFTWARE IS  SUBJECT TO CHANGE WITHOUT NOTICE   *
0000  ;*  AND  SHOULD  NOT  BE  CONSTRUED AS  A COMMITMENT BY DIGITAL EQUIPMENT   *
0000  ;*  CORPORATION.                                                            *
0000  ;*                                                                          *
0000  ;*  DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE  OR  RELIABILITY OF ITS   *
0000  ;*  SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.                 *
0000  ;*                                                                          *
0000  ;*                                                                          *
0000  ;****************************************************************************
0000  ;
0000  ;
0000  ; FACILITY: Language support library - user callable
0000  ;++
0000  ; ABSTRACT:
0000  ;
0000  ;          1)  Double base to double power.
0000  ;          2)  Floating base to double power.
0000  ;          3)  Double base to floating power.
0000  ;
0000  ;          Floating overflow can occur
0000  ;          Undefined exponentiation can occur if:
0000  ;                  1)  Negative base
0000  ;                  2)  0 base and power is 0 or negative.
0000  ;
0000  ;
0000  ;
0000  ;--
0000  ;
0000  ; VERSION: 2
0000  ;
0000  ; HISTORY:
0000  ;
0000  ; AUTHOR:
0000  ;          Bob Hanek, 3-Mar-83: Version 2
0000  ;
0000  ; MODIFIED BY:
0000  ;
0000  ;
0000  ;
```

```
0000           .SBTTL  HISTORY         ; Detailed current edit history
0000
0000
0000   ; Edit history for Version 2 of OTS$POWDD
0000   ;
0000   ; 2-001 Implemented new algorithm. RNH 3-Mar-83
0000   ; 2-002 The code would overflow when underflow was the proper result. This
0000   ;       error arouse from the ADDF3 #^X4DC0, R2, R6 code, which rounded the
0000   ;       result so that when   SUBF  #^X4DC0, R6  was performed the value in
0000   ;       R6 was greater than R2, causing R0 to be positive when it should have
0000   ;       been negative. The bugfix was to change the test for overflow from
0000   ;       R0 to R6 (both of these should be of the same sign). JCW 13-MAY-83.
0000   ; 2-003 Change INDEX table to be local instead of global.  Also change
0000   ;       the instruction format from INDEX(Rx) to INDEX[Rx] to avoid linker
0000   ;       errors.  LEB 25-May-1983
0000   ; 2-004 Change remaining table references to use [Rx] instead of (Rx).
0000   ;       Make A1_TABLE and A2_TABLE local symbols instead of globals.
0000   ;       LEB 26-May-1983
0000   ; 2-005 Change leftover table reference to use [Rx] instead of (Rx).
0000   ;       LEB 29-May-1983
0000   ; 2-006 Added two ROTL #-3,Rx,Rx instructions to scale the value of Rx back
0000   ;       from 'index*2^3' to 'index' before A_TABLE[Rx] is referenced. The
0000   ;       INDEX was not scaled back to yield values of 'index' instead of
0000   ;       'index*2^3' because the mathematics of the code used does need the
0000   ;       value of index*2^3 in several computations. JCW 7-Jun-1983
0000   ; 2-007 Corrected a bug involving a SYS_F_FLTOVF_F error during a MULD R6, R2.
0000   ;       Code was added to see if a MTH overflow message or a zero should be
0000   ;       returned.  JCW 19-Jan-1984
0000   ;
0000   ;
```

```
                   0000          .SBTTL  DECLARATIONS
                   0000
                   0000  ;
                   0000  ; INCLUDE FILES:
                   0000  ;
                   0000  ;
                   0000
                   0000  ;
                   0000  ; EXTERNAL SYMBOLS:
                   0000  ;
                   0000          .DSABL  GBL
                   0000          .EXTRN  MTH$$SIGNAL             ; Math error routine
                   0000          .EXTRN  MTH$K_FLOOVEMAT         ; Floating overflow code
                   0000          .EXTRN  MTH$K_FLOUNDMAT         ; Floating underflow code
                   0000          .EXTRN  MTH$K_UNDEXP            ; Undefined exponentiation code
                   0000
                   0000  ;
                   0000  ; MACROS:
                   0000  ;
                   0000          $SFDEF                          ; Define stack frame symbols
                   0000  ;
                   0000  ; EQUATED SYMBOLS:
                   0000  ;
         00000004  0000          base    = 4                     ; base input formal - by-value
         0000000C  0000          exp     = 12                    ; exponent input formal - by-value
         00000008  0000          fexp    = 8                     ; exponent when base is floating
         000007FC  0000          ACMASK  = ^M< R2, R3, R4, R5, R6, R7, R8, R9, R10>
                   0000                                          ; register saving mask
```

```
                          0000       ;
                          0000       ; PSECT DECLARATIONS:
                          0000       ;
                      00000000              .PSECT  _OTS$CODE       PIC,SHR,QUAD,EXE,NOWRT
                          0000                                      ; program section for OTS$ code
                          0000       ;
                          0000       ; CONSTANTS:
                          0000       ;
                          0000
                          0000       ;
                          0000       ; The INDEX table gives the appropriate byte offset into the A1 and A2 Tables.
                          0000       ;
                          0000
 08 08 08 08 08 00 00 00  0000       INDEX:  .BYTE   ^X00, ^X00, ^X00, ^X08, ^X08, ^X08, ^X08, ^X08
 18 10 10 10 10 10 10 08  0008               .BYTE   ^X08, ^X10, ^X10, ^X10, ^X10, ^X10, ^X10, ^X18
 20 20 20 18 18 18 18 18  0010               .BYTE   ^X18, ^X18, ^X18, ^X18, ^X18, ^X20, ^X20, ^X20
 28 28 28 28 20 20 20 20  0018               .BYTE   ^X20, ^X20, ^X20, ^X20, ^X28, ^X28, ^X28, ^X28
 30 30 30 30 30 30 28 28  0020               .BYTE   ^X28, ^X28, ^X30, ^X30, ^X30, ^X30, ^X30, ^X30
 38 38 38 38 38 38 30 30  0028               .BYTE   ^X30, ^X30, ^X38, ^X38, ^X38, ^X38, ^X38, ^X38
 40 40 40 40 40 40 40 38  0030               .BYTE   ^X38, ^X40, ^X40, ^X40, ^X40, ^X40, ^X40, ^X40
 48 48 48 48 48 48 48 40  0038               .BYTE   ^X40, ^X48, ^X48, ^X48, ^X48, ^X48, ^X48, ^X48
 50 50 50 50 50 50 50 48  0040               .BYTE   ^X48, ^X50, ^X50, ^X50, ^X50, ^X50, ^X50, ^X50
 58 58 58 58 58 58 50 50  0048               .BYTE   ^X50, ^X50, ^X58, ^X58, ^X58, ^X58, ^X58, ^X58
 60 60 60 60 60 58 58 58  0050               .BYTE   ^X58, ^X58, ^X58, ^X60, ^X60, ^X60, ^X60, ^X60
 68 68 68 68 60 60 60 60  0058               .BYTE   ^X60, ^X60, ^X60, ^X60, ^X68, ^X68, ^X68, ^X68
 70 70 68 68 68 68 68 68  0060               .BYTE   ^X68, ^X68, ^X68, ^X68, ^X68, ^X68, ^X70, ^X70
 70 70 70 70 70 70 70 70  0068               .BYTE   ^X70, ^X70, ^X70, ^X70, ^X70, ^X70, ^X70, ^X70
 78 78 78 78 78 78 78 78  0070               .BYTE   ^X78, ^X78, ^X78, ^X78, ^X78, ^X78, ^X78, ^X78
 80 80 80 80 80 78 78 78  0078               .BYTE   ^X78, ^X78, ^X78, ^X80, ^X80, ^X80, ^X80, ^X80
                          0080
                          0080
                          0080               .ALIGN  QUAD
                          0080
                          0080       ;
                          0080       ; For k = 0, 1, ..., 16, the k-th entry of the A1 table is 2^(k/16) rounded
                          0080       ; to 56 bits and the k-th entry of the A2 table 2^(k/16) - A1_TABLE(k)
                          0080       ; rounded to 56 bits.
                          0080       ;
                          0080
                          0080       A1_TABLE:
         00000000 00004080  0080             .QUAD   ^X0000000000004080
         487B67CC AAC34085  0088             .QUAD   ^X487B67CCAAC34085
         8BD7E3EA 95C1408B  0090             .QUAD   ^X8BD7E3EA95C1408B
         11C373AB C3D34091  0098             .QUAD   ^X11C373ABC3D34091
         B8A9518D 37F04098  00A0             .QUAD   ^XB8A9518D37F04098
         A1126091 F532409E  00A8             .QUAD   ^XA1126091F532409E
         5139A9B1 FED640A5  00B0             .QUAD   ^X5139A9B1FED640A5
         A14BEA42 583E40AD  00B8             .QUAD   ^XA14BEA42583E40AD
         DE6533F9 04F340B5  00C0             .QUAD   ^XDE6533F904F340B5
         0C379F58 08A340BD  00C8             .QUAD   ^X0C379F5808A340BD
         06DB1155 672A40C5  00D0             .QUAD   ^X06DB1155672A40C5
         8481151F 248C40CE  00D8             .QUAD   ^X8481151F248C40CE
         9D6BCAD6 44FC40D7  00E0             .QUAD   ^X9D6BCAD644FC40D7
         94E1EC2A CCDE40E0  00E8             .QUAD   ^X94E1EC2ACCDE40E0
         2439E7DD C0C640EA  00F0             .QUAD   ^X2439E7DDC0C640EA
         86CC1524 257D40F5  00F8             .QUAD   ^X86CC1524257D40F5
         00000000 00004100  0100             .QUAD   ^X0000000000004100
```

```
                           0108
                           0108  A2_TABLE:
00000000 00000000          0108            .QUAD   ^X0000000000000000
0AA6DC61 2E4A2326          0110            .QUAD   ^X0AA6DC612E4A2326
3AD6EBC5 20DCA348          0118            .QUAD   ^X3AD6EBC520DCA348
B96B6382 3F5B23D8          0120            .QUAD   ^XB96B63823F5B23D8
5C864630 8D5A245E          0128            .QUAD   ^X5C8646308D5A245E
288CC1EC BEDDA424          0130            .QUAD   ^X288CC1ECBEDDA424
477C04EF 1A14A32F          0138            .QUAD   ^X477C04EF1A14A32F
29975CDC D9FDA3E6          0140            .QUAD   ^X29975CDCD9FDA3E6
156BEC99 4D04A477          0148            .QUAD   ^X156BEC994D04A477
25E1C093 AEFDA402          0150            .QUAD   ^X25E1C093AEFDA402
6C8ABCDA 0754A38B          0158            .QUAD   ^X6C8ABCDA0754A38B
506AE383 EE53A360          0160            .QUAD   ^X506AE383EE53A360
FD2C4466 6597A2BC          0168            .QUAD   ^XFD2C44666597A2BC
6E9AA824 32C42308          0170            .QUAD   ^X6E9AA82432C42308
C1B9D6CD 40B523BB          0178            .QUAD   ^XC1B9D6CD40B523BB
B66031EB EE7423B1          0180            .QUAD   ^XB66031EBEE7423B1
00000000 00000000          0188            .QUAD   ^X0000000000000000
                           0190
00000000 00002500          0190  TWO_M55:.QUAD   ^X0000000000002500
                           0198
17F1295C AA3B44B8          0198  C:      .QUAD   ^X17F1295CAA3B44B8
00002000 AA3B44B8          01A0  C1:     .QUAD   ^X00002000AA3B44B8
E8800BBB C17F3695          01A8  C2:     .QUAD   ^XE8800BBBC17F3695
                           01B0
973BB7EA 01121CC2          01B0  LOGTAB: .QUAD   ^X973BB7EA01121CC2
2742C3D0 CA802581          01B8            .QUAD   ^X2742C3D0CA802581
F145FD62 19A02E3D          01C0            .QUAD   ^XF145FD6219A02E3D
1DA4FD64 FE9F3723          01C8            .QUAD   ^X1DA4FD64FE9F3723
00000000 00000000          01D0            .QUAD   ^X0000000000000000
         00000005          01D8  LOGLEN = <.-LOGTAB>/8
                           01D8
                           01D8
                           01D8  EXPTAB:
0D2F11A3 E6F41FFF          01D8            .QUAD   ^X0D2F11A3E6F41FFF        ;    0.27094695278739704E-19
2D1FB089 85242521          01E0            .QUAD   ^X2D1FB08985242521        ;    0.35024086731698588E-16
6DC43BFA C3FF2A2E          01E8            .QUAD   ^X6DC43BFAC3FF2A2E        ;    0.38805761563274748E-13
D9857D84 955B2F1D          01F0            .QUAD   ^XD9857D84955B2F1D        ;    0.35830323049894278E-10
0627B825 584633E3          01F8            .QUAD   ^X0627B825584633E3        ;    0.26466421444330986E-07
2C9CFC16 FDEF3875          0200            .QUAD   ^X2C9CFC16FDEF3875        ;    0.14662262387640433E-04
CF7AF7D1 72173CB1          0208            .QUAD   ^XCF7AF7D172173CB1        ;    0.54152123481245728E-02
00000000 00000000          0210            .QUAD   ^X0000000000000000        ;    0.0
         00000008          0218  EXPLEN = <.-EXPTAB>/8
                           0218
```

```
0218        .SBTTL  OTS$POWDD - DOUBLE to DOUBLE giving DOUBLE result
0218
0218  ;++
0218  ; FUNCTIONAL DESCRIPTION:
0218  ;
0218  ;       OTS$POWDR - DOUBLE result = DOUBLE base ** FLOATING exponent
0218  ;       OTS$POWRD - DOUBLE result = FLOATING base ** DOUBLE exponent
0218  ;       OTS$POWDD - DOUBLE result = DOUBLE base ** DOUBLE exponent
0218  ;
0218  ;       The DOUBLE result is given by:
0218  ;
0218  ;       base    exponent        result
0218  ;       ----    --------        ------
0218  ;
0218  ;       = 0     > 0             0.0
0218  ;       = 0     = 0             Undefined Exponentiation
0218  ;       = 0     < 0             Undefined Exponentiation
0218  ;
0218  ;       < 0     any             Undefined Exponentiation
0218  ;
0218  ;       > 0     > 0             2^(exp * LOG2(base))
0218  ;       > 0     = 0             1.0
0218  ;       > 0     < 0             2^(exp * LOG2(base))
0218  ;
0218  ;
0218  ;       Floating Overflow and Underflow can occur.
0218  ;       Undefined Exponentiation can occur if:
0218  ;               1)  base is 0 and exponent is 0 or negative
0218  ;               2)  base is negative
0218  ;
0218  ; The basic approach to computing x**y as 2^[y*log2(x)] is the following:
0218  ;
0218  ;       Step 1: Compute log2(x) to sufficient precision to guarantee an
0218  ;               accurate final result (see below.)
0218  ;       Step 2: Compute y*log2(x) to at least the accuracy that log2(x)
0218  ;               was computed.
0218  ;       Step 3: Evaluate 2^[y*log2(x)] accurate to the precision of the
0218  ;               datatype in question.
0218  ;
0218  ; To determine the accuracy to which log2(x) must be computed to, write
0218  ; y*log2(x) as I + h, where I is the integer closest to y*log2(x), and
0218  ; h = y*log2(x) - I (Note that |h| =< 1/2.)  Then
0218  ;
0218  ;               2^[y*log2(x)] = 2^(I + h) = (2^I)*(2^h).
0218  ;
0218  ; Since the factor 2^I can be incorporated into the final result by an integer
0218  ; addition to the exponent field, we can assume that the multiplication by
0218  ; 2^I incurs no error.  Thus the total error in the final result is determined
0218  ; by how accurately 2^h can be computed.  If the final result has p fraction
0218  ; bits, we would like h to have at least p good bits.  In fact it would be
0218  ; nice if h had a few extra guard bits, say 4.  Consequently, we would like
0218  ; h to be accurate to p + 4 bits.
0218  ;
0218  ; Let e be the number of bits allocated to the exponent field of the data type
0218  ; in question.  If I requires more that e bits to represent, then overflow or
0218  ; underflow will occur.  Therefore if the product y*log2(x) has e + p + 4 good
0218  ; bits, the final result will be accurate.  This requires that log2(x) be
```

```
0218  ; computed to at least p + e + 4 bits.
0218  ;
0218  ; Since log2(x) must be computed to more bits of precision than is available
0218  ; in the base data type, either the next level of precision or multi-precision
0218  ; arithmetic must be used.  We begin by writing
0218  ;
0218  ;
0218  ;                               \---
0218  ;                                          2n+1
0218  ;         log2(x) = log2(b) +  >  c(2n+1)*z',
0218  ;                              /
0218  ;                              n=0
0218  ;
0218  ; Where c(1) = 1, and z' = (2/ln2)[(z-b)/(z+b)].  Hence
0218  ;
0218  ;
0218  ;                                    \---
0218  ;                                               2n+1
0218  ;         log2(x) = log2(b) +  z' +  >  c(2n+1)*z'
0218  ;                                   /
0218  ;                                   n=1
0218  ;
0218  ;                 = log2(b) + z' + p(z').
0218  ;
0218  ; Note that if p(z') is computed to p bits, and log2(b) + z' is computed
0218  ; to p+e+4 bits and overhangs p(z') by e+4 bits, the required accuracy will
0218  ; be achieved.  Consequently, the essential tricks, are to pick b such that
0218  ; the overhang can be achieved and to compute log2(b) + z' to p + e + 4 bits.
0218  ;
0218  ;
0218  ; CALLING SEQUENCE:
0218  ;
0218  ;         power.wd.v = OTS$POWDR (base.rd.v, exponent.rf.v)
0218  ;         power.wd.v = OTS$POWRD (base.rf.v, exponent.rd.v)
0218  ;         power.wd.v = OTS$POWDD (base.rd.v, exponent.rd.v)
0218  ;
0218  ; INPUT PARAMETERS:
0218  ;         Base and exponent parameters are call by value
0218  ;
0218  ; IMPLICIT INPUTS:
0218  ;         none
0218  ;
0218  ; OUTPUT PARAMETERS:
0218  ;         none
0218  ;
0218  ; IMPLICIT OUTPUTS:
0218  ;         none
0218  ;
0218  ; FUNCTIONAL VALUE:
0218  ;         OTS$POWDR - DOUBLE base ** FLOATING power
0218  ;         OTS$POWRD - FLOATING base ** DOUBLE power
0218  ;         OTS$POWDD - DOUBLE base ** DOUBLE power
0218  ;
0218  ; SIDE EFFECTS:
0218  ;
0218  ;         SIGNALs MTH$K_FLOOVEMAT if floating overflow.
0218  ;         SIGNALs MTH$K_FLOUNDMAT if floating underflow.
0218  ;         SIGNALs MTH$_UNDEXP (82 = ' UNDEFINED EXPONENTIATION') if
0218  ;                 1)  Base is 0 and exponent is 0 or negative
```

```
0218     ;               2)  base is negative
0218     ;
0218     ;
0218     ;--
```

```
                  07FC  0218  339          .ENTRY  OTS$POWDR, ACMASK           ;
                        021A  340
         5E    14   C2  021A  341          SUBL    #20, SP                     ; Allocate 5 longwords on the stack
         6E    02   D0  021D  342          MOVL    #2, (SP)                    ; Set number of arguments equal to 2
      04 AE  04 AC  70  0220  343          MOVD    base(AP), base(SP)          ; Move base to stack
      0C AE  0C AC  56  0225  344          CVTFD   exp(AP), exp(SP)            ; convert FLOATING exponent to DOUBLE
 0000024C'EF   6E   FA  022A  345          CALLG   (SP), OTS$POWDD             ; Call OTS$POWDD
                    04  0231  346          RET
                        0232  347
                        0232  348
                  07FC  0232  349          .ENTRY  OTS$POWRD, ACMASK           ;
                        0234  350
         5E    14   C2  0234  351          SUBL    #20, SP                     ; Allocate 5 longwords on the stack
         6E    02   D0  0237  352          MOVL    #2, (SP)                    ; Set number of arguments equal to 2
      04 AE  04 AC  56  023A  353          CVTFD   base(AP), base(SP)          ; convert FLOATING base to DOUBLE
      0C AE  08 AC  70  023F  354          MOVD    fexp(AP), exp(SP)           ; Move exponent to stack
 0000024C'EF   6E   FA  0244  355          CALLG   (SP), OTS$POWDD             ; Call OTS$POWDD
                    04  024B  356          RET
                        024C  357
```

```
                        024C
                07FC    024C            .ENTRY  OTS$POWDD, ACMASK       ;
                        024E
                        024E    ; Move x to R0/R1.  If x < 0, or x = 0 and y =< 0, return ''UNDEFINED
                        024E    ; EXPONENTIATION'' error condition, otherwise attempt to compute x**y
                        024E    ;
                        024E
                        024E    GET_BASE:
     50    04 AC    70  024E            MOVD    base(AP), R0                    ; R0/R1 <-- x
                        0252    COMMON:
           1B       14  0252            BGTR    DEFINED                         ; If x > 0 attempt to compute x**y
           06       19  0254            BLSS    UNDEFINED                       ; Branch to error code for x < 0
        0C AC       73  0256            TSTD    exp(AP)                         ; Check sign of y (Note that x = 0)
           01       15  0259            BLEQ    UNDEFINED                       ; Branch to error condition if y =< 0
                        025B
                        025B    ;
                        025B    ; If processing continues here, this implies that x = 0 and y > 0.  Return
                        025B    ; x**y = 0
                        025B    ;
                        025B
                    04  025B            RET                                     ; Return
                        025C
                        025C    ;
                        025C    ; If processing continues here, this implies that an undefined exponentiation
                        025C    ; was attempted.  Signal error and return
                        025C    ;
                        025C
                        025C    UNDEFINED:
     50    8000 8F  3C  025C            MOVZWL  #^X8000, R0                     ; R0/R1 <-- Reserved operand
               51   D4  0261            CLRL    R1
        7E  00'8F   9A  0263            MOVZBL  #MTH$K_UNDEXP, -(SP)            ; Put error code on stack
  00000000'GF  01   FB  0267            CALLS   #1, G^MTH$$SIGNAL               ; Convert error number to 32 bit
                        026E                                                    ;    condition code and signal error.
                        026E                                                    ;    NOTE:  Second argument is not re-
                        026E                                                    ;    quired since there is no JSB entry.
                    04  026E            RET                                     ; Return
                        026F
                        026F    ;
                        026F    ; If processing continues here will attempt to compute x**y as 2^[y*log2(x)].
                        026F    ; We begin by determining k and f such that x = 2^k*f, where 1 =< f < 2.
                        026F    ;
                        026F
                        026F    DEFINED:
 54   50  FFFF807F 8F   CB  026F        BICL3   #^XFFFF807F, R0, R4             ; R4 <-- 2^7*(biased exponent of x)
      54  00004080 8F   C2  0277        SUBL    #^X4080, R4                     ; R4 <-- 2^7*k = 2^7*(exponent of x - 1)
              50   54   C2  027E        SUBL    R4, R0                          ; R0 <-- f = 2*(fraction field of x)
                        0281
                        0281    ;
                        0281    ; We are now ready to compute log2(x).  This computation is based on the
                        0281    ; following identity:
                        0281    ;
                        0281    ;                                   2   \---   1                            f-a
                        0281    ;      log2(2^k*f) = k + log2(a) + ----- >     ---- z^(2j+1), where z = -----.
                        0281    ;                                  ln(2)/____  2j+1                         f+a
                        0281    ;
                        0281    ; We begin by determining a as b^i, where b = 2^(1/16) and i is
                        0281    ; between 0 and 16 inclusive.  Specifically i is chosen by table look-up
```

```
                              0281  ; in such a fashion as to minimize the magnitude of z.  Since log2(a) = i/16
                              0281  ; we may write
                              0281  ;
                              0281  ;                 log2(x) = k + i/16 + z*p(z*z).
                              0281  ;
                              0281
                              0281  EVAL_LOG2:
 5A    50   FFFFFF80 8F   CB  0281          BICL3   #^XFFFFFF80, R0, R10     ; R10 <-- index to INDEX table
       5A   FD72 CF4A     90  0289          MOVB    INDEX[R10], R10          ; R10 <-- i*2^3
       7E    54    5A     C1  028F          ADDL3   R10, R4, -(SP)           ; SP -->  2^7*(k + i/16)
   5A  5A    FD  8F       9C  0293          ROTL    #-3, R10, R10            ; R10 <-- i
                              0298                                           ; R10 will be multiplied by 2^3 by
                              0298                                           ;   table references like the line below.
                              0298                                           ;   The linker will cause an error if
                              0298                                           ;   () are used instead of [] for these
                              0298                                           ;   table references.
                              0298  ;
                              0298  ; We proceed by computing z = (f-a)/(f+a).  In order to insure the accuracy of
                              0298  ; the final result, it is necessary to compute z to at least 68 bits. Since no
                              0298  ; back up data type is available, we must compute z in two parts: z = z1 + z2,
                              0298  ; where z1 is the high 24 bits of z and z2 is the low 56 bits of z.  Further,
                              0298  ; to obtain the desired accuracy it is necessary to work with a = a1 + a2,
                              0298  ; where a1 and a2 are the high and low 56 bits respectively of 'a'.  We begin
                              0298  ; computing (in single precision)
                              0298  ;
                              0298  ;                       z1 = (f - a1)/(f + a1)
                              0298  ;
                              0298  ; Note that f-a1 can be computed exactly in 56 bits, but f+a1 may require 57
                              0298  ; bits. The 57 bit can be determined by the exclusive or of the low bits of
                              0298  ; f and a1.
                              0298  ;
                              0298
       58   FDE3 CF4A     7D  0298          MOVQ    A1_TABLE[R10], R8        ; R8/R9 <-- a1
       7E    51    59     CD  029E          XORL3   R9, R1, -(SP)            ; SP --> XOR of low bits of a1 and x
                              02A2                                           ;     (This will used to determine the
                              02A2                                           ;      57 bit of f+a1.)
       52    50    58     61  02A2          ADDD3   R8, R0, R2               ; R2/R3 <-- f + a1 (rounded)
             50    58     62  02A6          SUBD    R8, R0                   ; R0/R1 <-- f - a1 (exact)
       58    50    52     47  02A9          DIVF3   R2, R0, R8               ; R8 <-- z1 (single)
             59    00     D0  02AD          MOVL    #0, R9                   ; R8/R9 <-- z1 (double)
                              02B0  ;
                              02B0  ; To compute z2 we note
                              02B0  ;
                              02B0  ;       z = z1 + z2 = (f - a1 - a2)/(f + a1 + a2)
                              02B0  ;
                              02B0  ;           ==>  z2 = (f - a1 - a2)/(f + a1 + a2) - z1
                              02B0  ;
                              02B0  ; Now let v = f + a1 + a2 = v1 + v2, where v1 and v2 are the high 24 and low
                              02B0  ; 56 bits of v respectively.  Then
                              02B0  ;
                              02B0  ;                 z2 = [(f - a1 - z1*v1) - (a2 + z1*v2)]/v
                              02B0  ;
                              02B0  ; We begin by computing v1 and f - a1 - z1*v1
                              02B0  ;
                              02B0
             54    52     D0  02B0          MOVL    R2, R4                   ; R4 <-- high longword of f + a1
```

```
            55   00  D0 02B3            MOVL    #0, R5                  ; R4/R5 <-- v1
       56   52   54  63 02B6            SUBD3   R4, R2, R6              ; R6/R7 <-- f + a1 - v1 (exact)
            54   58  64 02BA            MULD    R8, R4                  ; R4/R5 <-- z1*v1 (exact)
            50   54  62 02BD            SUBD    R4, R0                  ; R0/R1 <-- f - a1 - z1*v1 (exact)
                        02C0    ;
                        02C0    ; Compute v2 and a2 + a1*v2
                        02C0    ;
      54   FE43 CF4A 7D 02C0            MOVQ    A2_TABLE[R10], R4       ; R4/R5 <-- a2
  6E  FFFEFFFF    8F CA 02C6            BICL    #^XFFFEFFFF, (SP)       ; Check if w was rounded
                  05 13 02CD            BEQL    1$                      ; Branch if not rounded
       56   FEBD  CF 62 02CF            SUBD    TWO_M55, R6             ; Correct for rounding error (exact)
            56    54 60 02D4    1$:     ADDD    R4, R6                  ; R6/R7 <-- v2
            56    58 64 02D7            MULD    R8, R6                  ; R6/R7 <-- z1*v2
            56    54 60 02DA            ADDD    R4, R6                  ; R6/R7 <-- a2 + z1*v2
                        02DD    ;
                        02DD    ; Compute z2
                        02DD    ;
            50    56 62 02DD            SUBD    R6, R0                  ; R0/R1 <-- (f-a1-z1*v1)-(a2-z1*v2)
            50    52 66 02E0            DIVD    R2, R0                  ; R0/R1 <-- z2
                        02E3    ;
                        02E3    ; The next step is to compute log2(x) accurate to at least 68 bits.  This is
                        02E3    ; accomplished as follows, let
                        02E3    ;
                        02E3    ;               w = 2^7*log2(x)
                        02E3    ;                 = (2^7)[k + i/16 + z*p(z*z)]
                        02E3    ;                 = 2^7*(k + i/16) + (2^7)*z*(c0 + c2*z^2 + ... + c10*z^10)
                        02E3    ;                 = [2^7*(k + i/16) + z'] + z'(d2*z'^2 + ... + d10*z'^10)
                        02E3    ;                 = [2^7*(k + i/16) + z'] + z'*q(z'*z')
                        02E3    ;                 = w1 + w2
                        02E3    ;
                        02E3    ; where z' = (2^7*c0)*z and w1 and w2 are the high 24 and low 56 bits of w
                        02E3    ; respectively.  Note that the choice of 'a' used in computing z, guarantees
                        02E3    ; that z' overhangs z'*q(z'*z') by at least 13 bits.  Hence, if w is computed
                        02E3    ; as w1 + w2, the necessary 68 bits of accuracy can be obtained. c0=2^6/ln(2).
                        02E3    ;
                        02E3    ; We begin by defining
                        02E3    ;
                        02E3    ;               c  = high 56 bits of (2^7*c0)
                        02E3    ;               c1 = high 28 bits of (2^7*c0)
                        02E3    ;               c2 = low 56 bits of (2^7*c0)
                        02E3    ; then
                        02E3    ;               z' = (z1 + z2)*(c1 + c2)
                        02E3    ;                  = z1*c1 + z1*c2 + z2*c.
                        02E3    ;
                        02E3    ;
  54   58   FEB9  CF 65 02E3            MULD3   C1, R8, R4              ; R4/R5 <-- c1*z1
       58   FEBB  CF 64 02E9            MULD    C2, R8                  ; R8/R9 <-- c2*z1
       50   FEA6  CF 64 02EE            MULD    C, R0                   ; R0/R1 <-- c*z2
            50    58 60 02F3            ADDD    R8, R0                  ; R0/R1 <-- c*z2 + c2*z1
       58   50    54 61 02F6            ADDD3   R4, R0, R8              ; R8/R9 <-- z'
                        02FA
                        02FA    ;
                        02FA    ; We proceed by letting
                        02FA    ;
                        02FA    ;               w1 = high 24 bits of 2^7*(k + i/16) + z1*c1
                        02FA    ; and
                        02FA    ;               w2' = {[2^7*(k + i/16) + z1*c1 - w1] + z1*c2} + z2*c.
```

```
                          02FA  ;
                          02FA  ;               ==>  2^7*(k + i/16) + z' = w1 + w2'.
                          02FA  ;
                          02FA  ;               ==>  w  = [2^7*(k + i/16) + z'] + z'*q(z'*z')
                          02FA  ;                       = w1 + w2' + z'*q(z'*z')
                          02FA  ;                       = w1 + w2,
                          02FA  ;
                          02FA  ; where w2 = w2' + z'*q(z'*z')
                          02FA  ;
                          02FA
      5A   04 AE      4E  02FA          CVTLF   4(SP), R10              ; R10 <-- 2^7(k + i/16)
   56   54    5A      41  02FE          ADDF3   R10, R4, R6             ; R6 <-- 2^7(k+i/16) + z1*c1 in single
        57    00      D0  0302          MOVL    #0, R7                  ; R6/R7 <-- w1
   52   56    5A      43  0305          SUBF3   R10, R6, R2             ; R2 <-- bits of z1*c1 included in w1
                          0309                                          ;     = -[2(k+i/16)+z1*c1 - w1]
        53    00      D0  0309          MOVL    #0, R3                  ; Convert R2 to double
        54    52      62  030C          SUBD    R2, R4                  ; [2^7(k+i/16)+z1*c1 - w1] + c1*z1
   6E   50    54      61  030F          ADDD3   R4, R0, (SP)            ; (SP) --> w2'
                          0313
                          0313  ;
                          0313  ; Compute w2
                          0313  ;
                          0313
   50   58    58      65  0313          MULD3   R8, R8, R0              ; R0/R1 <-- z'*z'
FE93 CF   04  50      75  0317          POLYD   R0, #LOGLEN-1, LOGTAB   ; R0/R1 <-- q(z'*z')
        50    58      64  031D          MULD    R8, R0                  ; R0/R1 <-- z'*q(z'*z')
        50    6E      60  0320          ADDD    (SP), R0                ; R0/R1 <-- w2
                          0323
                          0323  ;
                          0323  ; We now calculate y*log2(x) = (y1+y2)*(w1+w2) = y1*w1 + y2*w1 + y*w2, where
                          0323  ; y1 and y2 are the high and low 28 bits of y respectively.
                          0323  ;
                          0323
      54   0C AC      7D  0323          MOVQ    exp(AP), R4             ; R4/R5 <-- y
                          0327
                          0327  ;
                          0327  ; Test for the possibility of overflow in the computation of y*w1.
                          0327  ; This will occur if the exponent of y plus the exponent of w1 is greater
                          0327  ; than 127.
                          0327  ;
                          0327
 52   54   08    07   EF  0327          EXTZV   #7, #8, R4, R2          ; biased exp of y
      52   0080  8F   A2  032C          SUBW2   #^X80, R2               ; unbiased exp of y
 53   56   08    07   EF  0331          EXTZV   #7, #8, R6, R3          ; biased exp of w1
      53   0080  8F   A2  0336          SUBW2   #^X80, R3               ; unbiased exp of w1
           53    52   A0  033B          ADDW2   R2, R3                  ; unbiased exp of w1*y
      53   007F  8F   B1  033E          CMPW    #^X7F, R3               ; largest unbiased exp possible is 127
                 76   19  0343          BLSS    Y_TIMES_W1_OVER
           50    54   64  0345          MULD    R4, R0                  ; R0/R1 <-- y*w2
           52    54   D0  0348          MOVL    R4, R2                  ; R2 <-- high longword of y1
53   55  FFFF0FFF 8F  CB  034B          BICL3   #^XFFFF0FFF, R5, R3     ; R2/R3 <-- y1
           54    52   62  0353          SUBD    R2, R4                  ; R4/R5 <-- y2
           52    56   64  0356          MULD    R6, R2                  ; R2/R3 <-- y1*w1
           54    56   64  0359          MULD    R6, R4                  ; R4/R5 <-- y2*w1
                          035C
                          035C  ;
                          035C  ; The next step in computing 2^[y*log2(x)] is to write y*log2(x) as
```

```
                              035C      ;
                              035C      ;               y*log2(x) = I + j/16 + g/16,
                              035C      ;
                              035C      ; where I is an integer, j is an integer between 0 and 15 inclusive, and
                              035C      ; g is a fraction in the interval [-1/2, 1/2)
                              035C      ;
56  52   00004DC0 8F   41     035C              ADDF3   #^X4DC0, R2, R6        ; 3*2^5 is used in this truncation process
                              0364                                             ;  to avoid a possible normalization
                              0364                                             ;  that could occur if the number is neg
     56  00004DC0 8F   42     0364              SUBF    #^X4DC0, R6            ; R6 <-- 2^7(I + j/16) in single
                57     00 D0  036B              MOVL    #0, R7                 ; R6/R7 <-- 2^7(I + j/16) in double
                52     56 62  036E              SUBD    R6, R2                 ; R2/R3 <-- those bits of w1*y1 not
                              0371                                             ;             included in 2^7(I + j/16)
                54     52 60  0371              ADDD    R2, R4                 ;
                50     54 60  0374              ADDD    R4, R0                 ; R0/R1 <-- g
                57     56 49  0377              CVTFW   R6, R7                 ; R7 <-- 2^7*(I + j/16) in integer
                       35 1D  037A              BVS     EXCEPTION_1            ; Branch if |I| is too large
                              037C
                              037C      ;
                              037C      ; We can now compute
                              037C      ;
                              037C      ;       x**y = 2^[y*log2(x)] = 2^[I + j/16 + g/16]
                              037C      ;
                              037C      ;            = (2^I)*[A*(B+1)] = 2^I*[A + A*B], where
                              037C      ;
                              037C      ; A = 2^(j/16) is obtained by table look-up and B = 2^(g/16) - 1 is obtained
                              037C      ; by a Min/Max approximation.
                              037C      ;
   FE56 CF     07  50    75   037C              POLYD   R0, #EXPLEN-1, EXPTAB  ; R0 <-- B =  2^(g/16) - 1. (Chebyshev
                              0382                                             ;    polynomial approx. of degree 5)
52  57   FFFFFF80 8F     CB   0382              BICL3   #^XFFFFFF80, R7, R2    ; R2 <-- 2^3 * index into A1_TABLE
    52     52    FD 8F   9C   038A              ROTL    #-3, R2, R2            ; R2 <-- index into A1_TABLE
     50    FCEC CF42     64   038F              MULD    A1_TABLE[R2], R0       ; R0/R1 <-- A*B
     50    FD6E CF42     60   0395              ADDD    A2_TABLE[R2], R0       ; R0/R1 <-- A*B + A2
     50    FCE0 CF42     60   039B              ADDD    A1_TABLE[R2], R0       ; R0/R1 <-- 2^[(j+g)/16] = (A*B+A2)+A1
        57    007F 8F    AA   03A1              BICW    #^X7F, R7              ; R7 = 2^7*I
                50  57   A0   03A6              ADDW    R7, R0                 ; R0 <-- 2^I*2^[(j+g)/16]
        007F 8F     50   B1   03A9              CMPW    R0, #^X7F              ; test for over/underflow
                    07   15   03AE              BLEQ    EXCEPTION_2            ; see what exception is if neg or = 0
                         04   03B0      RETURN: RET                            ; otherwise return result in R0
                              03B1
                              03B1      ;
                              03B1      ; Handlers for software detected over/underflow conditions follow
                              03B1      ;
                              03B1
                              03B1      EXCEPTION_1:
                56       53   03B1              TSTF    R6                     ; if big ARG > 0 goto overflow
                1D       18   03B3              BGEQ    OVER                   ;    handler, otherwise go to
                08       11   03B5              BRB     UNDER                  ;    underflow handler
                              03B7
                              03B7      EXCEPTION_2:
                57       B5   03B7              TSTW    R7                     ; test sign of I; if I < 0
                17       18   03B9              BGEQ    OVER                   ; go to overflow handler
                              03BB
                              03BB      ;
```

```
                        03BB  ; y*w1 would have caused a hardware system floating overflow error. If y<0,
                        03BB  ; then we should return a result of 0 since result = 2^(y*(w1+w2)). Note,
                        03BB  ; y can not be zero.
                        03BB  ;
                        03BB
                        03BB  Y_TIMES_W1_OVER:
              54   73   03BB          TSTD    R4                          ; if y < 0 no overflow is needed
              13   14   03BD          BGTR    OVER                        ; overflow
                        03BF
                        03BF  ;
                        03BF  ; Underflow; if user has FU set, signal error.  Always return 0.0
                        03BF  ;
                        03BF
                        03BF  UNDER:
              50   7C   03BF          CLRQ    R0                          ; R0/R1 <-- 0
        0B 04 AD   06   E1   03C1     BBC     #6, SF$W_SAVE_PSW(FP), 2$
                        03C6                                              ; has user enabled floating underflow?
        7E  00'8F  9A   03C6          MOVZBL  #MTH$K_FLOUNDMAT, -(SP)     ; Put underflow code on stack.
  00000000'GF  01  FB   03CA          CALLS   #1, G^MTH$$SIGNAL           ; Convert to MTH$_FLOUNDMAT (32-bit
                        03D1                                              ; VAX-11  exception code) and
                        03D1                                              ; signal condition
                   04   03D1  2$:     RET                                 ; return
                        03D2
                        03D2  ;
                        03D2  ; Signal floating overflow, return reserved operand, -0.0
                        03D2  ;
                        03D2
                        03D2  OVER:                                       ; else process for overflow
        7E  00'8F  9A   03D2          MOVZBL  #MTH$K_FLOOVEMAT, -(SP)     ; Put overflow code on stack
     50  01  0F    79   03D6          ASHQ    #15, #1, R0                 ; R0 = result = reserved operand
                        03DA                                              ; -0.0. R0 will be copied to
                        03DA                                              ; signal mechanism vector (CHF$L_MCH_R0/R1)
                        03DA                                              ; so can be fixed up by any error
                        03DA                                              ; handler
  00000000'GF  01  FB   03DA          CALLS   #1, G^MTH$$SIGNAL           ; signal condition
                   04   03E1          RET                                 ; return - R0 restored from CHF$L_MCH_R0/R1
                        03E2
                        03E2          .END
```

OTS
VAX
Ass
The
916
The
380
11
Mac

_$2
148
The
MAC

```
A1_TABLE             00000080 R      02
A2_TABLE             00000108 R      02
ACMASK             = 000007FC
BASE               = 00000004
C                    00000198 R      02
C1                   000001A0 R      02
C2                   000001A8 R      02
COMMON               00000252 R      02
DEFINED              0000026F R      02
EVAL_LOG2            00000281 R      02
EXCEPTION_1          000003B1 R      02
EXCEPTION_2          000003B7 R      02
EXP                = 0000000C
EXPLEN             = 00000008
EXPTAB               000001D8 R      02
FEXP               = 00000008
GET_BASE             0000024E R      02
INDEX                00000000 R      02
LOGLEN             = 00000005
LOGTAB               000001B0 R      02
MTH$$SIGNAL          ********     X  00
MTH$K_FLOOVEMAT      ********     X  00
MTH$K_FLOUNDMAT      ********     X  00
MTH$K_UNDEXP         ********     X  00
OTS$POWDD            0000024C RG     02
OTS$POWDR            00000218 RG     02
OTS$POWRD            00000232 RG     02
OVER                 000003D2 R      02
RETURN               000003B0 R      02
SF$W_SAVE_PSW      = 00000004
TWO_M55              00000190 R      02
UNDEFINED            0000025C R      02
UNDER                000003BF R      02
Y_TIMES_W1_OVER      000003BB 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
_OTS$CODE           000003E2  (   994.)  02 (  2.)    PIC   USR   CON   REL   LCL   SHR   EXE   RD   NOWRT NOVEC QUAD

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

Phase                    Page faults    CPU Time       Elapsed Time
-----                    -----------    --------       ------------
Initialization                    32    00:00:00.10    00:00:00.63
Command processing               125    00:00:00.75    00:00:06.82
Pass 1                           134    00:00:02.35    00:00:06.76
Symbol table sort                  0    00:00:00.05    00:00:00.10
Pass 2                           130    00:00:01.51    00:00:04.55
Symbol table output                4    00:00:00.06    00:00:00.54
```

```
Psect synopsis output             1     00:00:00.02     00:00:00.02
Cross-reference output            0     00:00:00.00     00:00:00.00
Assembler run totals            428     00:00:04.85     00:00:19.43

The working set limit was 1200 pages.
12217 bytes (24 pages) of virtual memory were used to buffer the intermediate code.
There were 10 pages of symbol table space allocated to hold 62 non-local and 2 local symbols.
741 source lines were read in Pass 1, producing 21 object records in Pass 2.
9 pages of virtual memory were used to define 8 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$:OTSPOWDD/OBJ=OBJ$:OTSPOWDD MSRC$:MTHJACKET/UPDATE=(ENH$:MTHJACKET)+MSRC
```

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY

OTSMULCD
LIS

OTSPOWCGC
LIS

OTSDIVC
LIS

OTSPOWDD
LIS

OTSPOWCC
LIS

OTSPOWGG
LIS

OTSPOWGJ
LIS

OTSPOWCDJ
LIS

MTHTAN
LIS

MTHVECTOR
LIS

OTSDIVCG
LIS

OTSPOWCJ
LIS

OTSPOWDLU
LIS

MTHTANH
LIS

OTSMULCG
LIS

OTSPOWCGJ
LIS

OTSPOWDJ
LIS

OTSDIVCD
LIS

OTSPOWCDC
LIS