F 13

```
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
```

```
**FILE**ID**UVXPOWRR                                        G 16


UU      UU  VV      VV  XX      XX  PPPPPPPP      OOOOOO    WW      WW  RRRRRRRR    RRRRRRRR
UU      UU  VV      VV  XX      XX  PPPPPPPP      OOOOOO    WW      WW  RRRRRRRR    RRRRRRRR
UU      UU  VV      VV  XX      XX  PP      PP  OO      OO  WW      WW  RR      RR  RR      RR
UU      UU  VV      VV  XX      XX  PP      PP  OO      OO  WW      WW  RR      RR  RR      RR
UU      UU  VV      VV    XX  XX    PP      PP  OO      OO  WW      WW  RR      RR  RR      RR
UU      UU  VV      VV    XX  XX    PP      PP  OO      OO  WW      WW  RR      RR  RR      RR
UU      UU  VV      VV      XX      PPPPPPPP    OO      OO  WW      WW  RRRRRRRR    RRRRRRRR
UU      UU  VV      VV      XX      PPPPPPPP    OO      OO  WW      WW  RRRRRRRR    RRRRRRRR
UU      UU  VV      VV    XX  XX    PP          OO      OO  WW  WW  WW  RR  RR      RR  RR
UU      UU  VV      VV    XX  XX    PP          OO      OO  WW  WW  WW  RR  RR      RR  RR
UU      UU    VV  VV    XX      XX  PP          OO      OO  WWWW  WWWW  RR    RR    RR    RR       ....
UU      UU    VV  VV    XX      XX  PP          OO      OO  WWWW  WWWW  RR    RR    RR    RR       ....
UUUUUUUUUU      VV      XX      XX  PP            OOOOOO    WW      WW  RR      RR  RR      RR     ....
UUUUUUUUUU      VV      XX      XX  PP            OOOOOO    WW      WW  RR      RR  RR      RR     ....

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  UVX$POWRR - REAL ** REAL power routine
0000          .IDENT  /2-008/          ; File: OTSPOWRR.MAR  Edit: JCW2008
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    ;       REAL base to REAL 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    ; VERSION: 2
0000    ;
0000    ; HISTORY:
0000    ;
0000    ; AUTHOR:
0000    ;       Bob Hanek, 3-Mar-83: Version 2
0000    ;
0000    ; MODIFIED BY:
0000    ;
0000    ;       Jeffrey C. Wiener, 9-MAY-83: Version 2-002
0000    ;
```

```
0000          .SBTTL  HISTORY         ; Detailed current edit history
0000
0000
0000    ; Edit history for Version 2 of OTS$POWRR
0000    ;
0000    ; 2-001 New algorithm implemented. RNH 3-Mar-83
0000    ; 2-002 Since microVAX requires that F_floating point routines may only
0000    ;       be backed-up by G_floating point instructions, the previous version
0000    ;       of this routine has been modified to accomplish this requirement.
0000    ;       JCW 9-MAY-83.
0000    ; 2-003 Change INDEX table to be a local table rather than a GLOBAL table.
0000    ;       LEB 24-May-1983.
0000    ; 2-004 Change reference of INDEX(Rx) to be INDEX[Rx] to avoid linker
0000    ;       errors.  LEB 25-May-1983
0000    ; 2-005 Changed MTH$POWRR entry to OTS$POWRR entry. JCW 26-May-1983
0000    ; 2-006 Change reference of A_TABLE(Rx) to be A_TABLE[Rx].  LEB 26-May-1983
0000    ; 2-007 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    ; 2-008 Corrected two bugs. The first bug was the omission of an A_TABLE
0000    ;       entry for 2^(16/16). While doing this I also converted all the
0000    ;       A_TABLE entries so that they now represented rounded values, rather
0000    ;       than truncated values. This will increase accuracy. This was also
0000    ;       done for C0. The other bug involved a SYS_F_FLTOVF_F error during
0000    ;       a MULG2 R0, R2. Code was added to see if a MTH overflow message or a
0000    ;       zero should be returned. While examining the code I also noticed that
0000    ;       the accuracy could be increased by replacing some of the CVTGF's (which
0000    ;       round) by code that would cause the CVTGF to truncate. The last CVTGF
0000    ;       in the code was not fixed in this manner because at this level in the
0000    ;       routine you are ready to return your result, which is always rounded
0000    ;       off.  JCW 19-Jan-1984
0000    ;
```

```
                           0000        .SBTTL  DECLARATIONS
                           0000
                           0000   ;
                           0000   ; INCLUDE FILES:
                           0000   ;
                           0000
                           0000   ;
                           0000   ; EXTERNAL SYMBOLS:
                           0000   ;
                           0000        .DSABL  GBL
                           0000        .EXTRN  MTH$K_UNDEXP          ; Undefined exponentiation
                           0000        .EXTRN  MTH$K_FLOUNDMAT       ; Underflow
                           0000        .EXTRN  MTH$K_FLOOVEMAT       ; Overflow
                           0000        .EXTRN  MTH$$SIGNAL           ; Math error routine
                           0000   ;
                           0000   ; MACROS:
                           0000   ;
                           0000        $SFDEF                        ; Define stack frame symbols
                           0000
                           0000   ;
                           0000   ; EQUATED SYMBOLS:
                           0000   ;
                  00000004 0000        base    = 4                   ; base input formal - by-value
                  00000008 0000        exp     = 8                   ; exponent input formal - by-value
                  0000001C 0000        ACMASK  = ^M< R2, R3, R4>     ; register saving mask
                  384F43F6 0000        C2      = ^X384F43F6
                  C0234393 0000        C4      = ^XC0234393
                           0000   ;
                           0000   ; OWN STORAGE:
                           0000   ;
                           0000   ;    none
                           0000   ;
                           0000   ;
                           0000   ; PSECT DECLARATIONS:
                           0000   ;
                       00000000        .PSECT  _OTS$CODE      PIC,SHR,LONG,EXE,NOWRT
                           0000                                      ; program section for OTS$ code
                           0000   ;
                           0000   ; CONSTANTS:
                           0000   ;
                           0000
                           0000   ;
                           0000   ; The INDEX table gives the byte offset into the A_TABLE necessary to select
                           0000   ; the proper choice of 'a.'
                           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
      832D652B 15474097  0080  C0:     .QUAD   ^X832D652B15474097
                         0088
                         0088  ;
                         0088  ; The ith entry of the A_TABLE contains the value 2^(i/16)
                         0088  ;
                         0088
      00000000 00004010  0088  A_TABLE:.QUAD   ^X0000000000004010      ; 2^( 0/16)
      890F6CF9 B5584010  0090          .QUAD   ^X890F6CF9B5584010      ; 2^( 1/16)
      517B3C7D 72B84011  0098          .QUAD   ^X517B3C7D72B84011      ; 2^( 2/16)
      62386E75 387A4012  00A0          .QUAD   ^X62386E75387A4012      ; 2^( 3/16)
      B7150A31 06FE4013  00A8          .QUAD   ^XB7150A3106FE4013      ; 2^( 4/16)
      34224C12 DEA64013  00B0          .QUAD   ^X34224C12DEA64013      ; 2^( 5/16)
      2A27D536 BFDA4014  00B8          .QUAD   ^X2A27D536BFDA4014      ; 2^( 6/16)
      5429DD48 AB074015  00C0          .QUAD   ^X5429DD48AB074015      ; 2^( 7/16)
      3BCD667F A09E4016  00C8          .QUAD   ^X3BCD667FA09E4016      ; 2^( 8/16)
      018773EB A1144017  00D0          .QUAD   ^X018773EBA1144017      ; 2^( 9/16)
      A0DB422A ACE54018  00D8          .QUAD   ^XA0DB422AACE54018      ; 2^(10/16)
      F09082A3 C4914019  00E0          .QUAD   ^XF09082A3C4914019      ; 2^(11/16)
      D3AD995A E89F401A  00E8          .QUAD   ^XD3AD995AE89F401A      ; 2^(12/16)
      529CDD85 199B401C  00F0          .QUAD   ^X529CDD85199B401C      ; 2^(13/16)
      A487DCFB 5818401D  00F8          .QUAD   ^XA487DCFB58'8401D      ; 2^(14/16)
      90DAA2A4 A4AF401E  0100          .QUAD   ^X90DAA2A4A4AF401E      ; 2^(15/16)
      00000000 00004020  0108          .QUAD   ^X0000000000004020      ; 2^(16/16)
                         0110
               59FC33E3  0110  EXPTAB: .LONG   ^X59FC33E3
               00663876  0114          .LONG   ^X00663876
               72183CB1  0118          .LONG   ^X72183CB1
               9625B19D  011C          .LONG   ^X9625B19D
               00000004  0120  EXPLEN = <.-EXPTAB>/4
```

```
0120           .SBTTL  OTS$POWRR - REAL to REAL giving REAL result
0120
0120    ;++
0120    ; FUNCTIONAL DESCRIPTION:
0120    ;
0120    ;       OTS$POWRR - REAL result = REAL base ** REAL exponent
0120    ;
0120    ;       The REAL result is given by:
0120    ;
0120    ;       base    exponent        result
0120    ;       ----    --------        ------
0120    ;
0120    ;       = 0     > 0             0.0
0120    ;       = 0     = 0             Undefined Exponentiation
0120    ;       = 0     < 0             Undefined Exponentiation
0120    ;
0120    ;       < 0     any             Undefined Exponentiation
0120    ;
0120    ;       > 0     > 0             2^[exp*log2(base)]
0120    ;       > 0     = 0             1.0
0120    ;       > 0     < 0             2^[exp*log2(base)]
0120    ;
0120    ;
0120    ;       Floating Overflow and Underflow can occur.
0120    ;       Undefined Exponentiation can occur if:
0120    ;               1)  base is 0 and exponent is 0 or negative
0120    ;               2)  base is negative
0120    ;
0120    ; The basic approach to computing x**y as 2^[y*log2(x)] is the following:
0120    ;
0120    ;       Step 1: Compute log2(x) to sufficient precision to guarantee an
0120    ;               accurate final result (see below.)
0120    ;       Step 2: Compute y*log2(x) to at least the accuracy that log2(x)
0120    ;               was computed.
0120    ;       Step 3: Evaluate 2^[y*log2(x)] accurate to the precision of the
0120    ;               datatype in question.
0120    ;
0120    ; To determine the accuracy to which log2(x) must be computed to, write
0120    ; y*log2(x) as I + h, where I is the integer closest to y*log2(x), and
0120    ; h = y*log2(x) - I (Note that |h| =< 1/2.)  Then
0120    ;
0120    ;               2^[y*log2(x)] = 2^(I + h) = (2^I)*(2^h).
0120    ;
0120    ; Since the factor 2^I can be incorporated into the final result by an integer
0120    ; addition to the exponent field, we can assume that the multiplication by
0120    ; 2^I incurs no error.  Thus the total error in the final result is determined
0120    ; by how accurately 2^h can be computed.  If the final result has p fraction
0120    ; bits, we would like h to have at least p good bits.  In fact it would be
0120    ; nice if h had a few extra guard bits, say 4.  Consequently, we would like
0120    ; h to be accurate to p + 4 bits.
0120    ;
0120    ; Let e be the number of bits allocated to the exponent field of the data type
0120    ; in question.  If I requires more that e bits to represent, then overflow or
0120    ; underflow will occur.  Therefore if the product y*log2(x) has e + p + 4 good
0120    ; bits, the final result will be accurate.  This requires that log2(x) be
0120    ; computed to at least p + e + 4 bits.
0120    ;
```

```
0120 ; Since log2(x) must be computed to more bits of precision than is available
0120 ; in the base data type, either the next level of precision or multi-precision
0120 ; arithmetic must be used.  We begin by writing
0120 ;
0120 ;
0120 ;                                    ---
0120 ;                                    \          2n+1
0120 ;       log2(x) = log2(b) +           >  c(2n+1)*z',
0120 ;                                    /
0120 ;                                    n=0-
0120 ;
0120 ; Where c(1) = 1, and z' = (2/ln2)[(z-b)/(z+b)].  Hence
0120 ;
0120 ;
0120 ;                                          ---
0120 ;                                          \          2n+1
0120 ;       log2(x) = log2(b) +  z' +           >  c(2n+1)*z'
0120 ;                                          /
0120 ;                                          n=1--
0120 ;
0120 ;               = log2(b) + z' + p(z').
0120 ;
0120 ; Note that if p(z') is computed to p bits, and log2(b) + z' is computed
0120 ; to p+e+4 bits and overhangs p(z') by e+4 bits, the required accuracy will
0120 ; be achieved.  Consequently, the essential tricks, are to pick b such that
0120 ; the overhang can be achieved and to compute log2(b) + z' to p + e + 4 bits.
0120 ;
0120 ;
0120 ; CALLING SEQUENCE:
0120 ;
0120 ;       power.wf.v = OTS$POWRR (base.rf.v, exponent.rf.v)
0120 ;
0120 ; INPUT PARAMETERS:
0120 ;       Base and exponent parameters are call by value
0120 ;
0120 ; IMPLICIT INPUTS:
0120 ;       none
0120 ;
0120 ; OUTPUT PARAMETERS:
0120 ;       none
0120 ;
0120 ; IMPLICIT OUTPUTS:
0120 ;       none
0120 ;
0120 ; FUNCTIONAL VALUE:
0120 ;       OTS$POWRR - REAL base ** REAL power
0120 ;
0120 ; SIDE EFFECTS:
0120 ;
0120 ;       SIGNALs MTH$K_FLOOVEMAT if floating overflow.
0120 ;       SIGNALs MTH$K_FLOUNDMAT if floating underflow.
0120 ;       SIGNALs MTH$K_UNDEXP (82 = ' UNDEFINED EXPONENTIATION') if
0120 ;               1)  base is 0 and exponent is 0 or negative
0120 ;               2)  base is negative
0120 ;
0120 ;
0120 ;--
```

```
                      001C 0120         .ENTRY  OTS$POWRR, ACMASK       ; standard call-by-reference entry
                           0122                                         ; disable DV (and FU)
                           0122
                           0122   ;
                           0122   ; Move x to R0.  If x < 0, or x = 0 and y =< 0, return ''UNDEFINED
                           0122   ; EXPONENTIATION'' error condition, otherwise attempt to compute x**y
                           0122   ;
                           0122
         50    04 AC    50 0122         MOVF    base(AP), R0            ; R0 <-- x
                  1A    14 0126         BGTR    DEFINED                 ; If x > 0 attempt to compute x**y
                  07    19 0128         BLSS    UNDEFINED               ; Branch to error code for x < 0
         51    08 AC    50 012A         MOVF    exp(AP), R1             ; R1 <-- y (Note that x = 0)
                  01    15 012E         BLEQ    UNDEFINED               ; Branch to error condition if y =< 0
                           0130
                           0130   ;
                           0130   ; If processing continues here, this implies that x = 0 and y > 0.  Return
                           0130   ; with x**y = 0
                           0130   ;
                           0130
                        04 0130         RET                             ; Return
                           0131
                           0131   ;
                           0131   ; If processing continues here, this implies that an undefined exponentiation
                           0131   ; was attempted.  Signal error and return
                           0131   ;
                           0131
                           0131   UNDEFINED:
       50    8000 8F    3C 0131         MOVZWL  #^X8000, R0             ; R0 <-- Reserved operand
       7E      00'8F    9A 0136         MOVZBL  #MTH$K_UNDEXP, -(SP)    ; Put error code on stack
  00000000'GF      01   FB 013A         CALLS   #1, G^MTH$$SIGNAL       ; Convert error number to 32 bit
                           0141                                         ;   condition code and signal error.
                           0141                                         ;   NOTE:  Second argument is not re-
                           0141                                         ;   quired since there is no JSB entry.
                        04 0141         RET                             ; Return
                           0142
                           0142   ;
                           0142   ; If processing continues here will attempt to compute x**y as 2^[y*log2(x)].
                           0142   ; We begin by determining an integer k and a real mumber f such that x = 2^k*f,
                           0142   ; and 1 =< f < 2.
                           0142   ;
                           0142
                           0142   DEFINED:
  54   50   FFFF807F 8F    CB 0142         BICL3   #^XFFFF807F, R0, R4     ; R4 <-- 2^7*(biased exponent of x)
       54   00004080 8F    C2 014A         SUBL    #^X4080, R4             ; R4 <-- 2^7*k = 2^7*(exponent_of_x - 1)
                50   54    C2 0151         SUBL    R4, R0                  ; R0 <-- f = 2*(fraction field of x)
                           0154
                           0154   ;
                           0154   ; We are now ready to compute log2(x).  This computation is based on the
                           0154   ; following identity:
                           0154   ;
                           0154   ;                                    2  \---    1                       f-a
                           0154   ;  log2(2^k*f) = k + log2(a) + ----- >    ---- z^(2j+1), where z = -----.
                           0154   ;                                 ln(2)/____ 2j+1                       f+a
                           0154   ;
                           0154   ; We begin by determining a as b^i, where b = 2^(1/16) and i is between 0
                           0154   ; and 16 inclusive.  Specifically i is chosen by table look-up so that
                           0154   ; the magnitude of z is minimized.  Since log2(a) = i/16, we may write
```

```
                                0154  ;
                                0154  ;                 log2(2^k*f) = k + i/16 + z*p(z^2).
                                0154  ;
                                0154  ; Note that in order to insure an accurate result, log2(2^k*f) must be computed
                                0154  ; accurately to 36 bits.  This will require some double precision arithmetic.
                                0154  ;
                                0154
                                0154  EVAL_LOG2:
  52   50   FFFFFF80 8F    CB   0154          BICL3   #^XFFFFFF80, R0, R2     ; R2 <-- index to INDEX table
       52      FE9F CF42   90   015C          MOVB    INDEX[R2], R2           ; R2 <-- i*2^3
            54        52   C0   0162          ADDL    R2, R4                  ; R4 <-- 2^7*(k + i/16)
      52   52      FD 8F   9C   0165          ROTL    #-3, R2, R2             ; R2 <-- i
                                016A                                          ; R2 will be multiplied by 2^3 by
                                016A                                          ;  table references like the line below.
                                016A                                          ;  The linker will cause an error if
                                016A                                          ;  () are used instead of [] for these
                                016A                                          ;  table references.
            50        50 99FD   016A          CVTFG   R0, R0                  ; R0/R1 <-- f
   52   50   FF14 CF42 43FD     016E          SUBG3   A_TABLE[R2], R0, R2     ; R2/R3 <-- f - a (NOTE: result is
                                0176                                          ;    exact, i.e. no roundoff error)
            50        10   A0   0176          ADDW    #^X10, R0               ; R0/R1 <-- 2*f
            50        52 42FD   0179          SUBG2   R2, R0                  ; R0/R1 <-- f + a
            52        50 46FD   017D          DIVG2   R0, R2                  ; R2/R3 <-- z
                                0181
                                0181  ;
                                0181  ; Compute 2^7*z*p(z^2) = z*(c0* + c2*z^2 + c4*z^4), where the c's are chosen
                                0181  ; to minimize the absolute error of the approximation
                                0181  ;
                                0181
  7E   53   FFFF1FFF 8F    CB   0181          BICL3   #^XFFFF1FFF, R3, -(SP)  ; prepare to save R2 and to clear
            7E        52   D0   0189          MOVL    R2, -(SP)               ;  the rounding bit in order to
            7E     8E 33FD      018C          CVTGF   (SP)+, -(SP)            ;  to form a truncated CVTGF
       51   8E        6E   45   0190          MULF3   (SP), (SP)+, R1         ; R1 <-- z^2
  50   51   C0234393 8F    45   0194          MULF3   #C4, R1, R0             ; R0 <-- c4*z^2
       50   384F43F6 8F    40   019C          ADDF    #C2, R0                 ; R0 <-- c2 + c4*z^2
            50        51   44   01A3          MULF    R1, R0                  ; R0 <-- c2*z^2 + c4*z^4
            50        50 99FD   01A6          CVTFG   R0, R0                  ; R0/R1 <-- c2*z^2 + c4*z^4
       50   FED1 CF 40FD        01AA          ADDG2   C0, R0                  ; R0/R1 <-- c0 + c2*z^2 + c4*z^4
            52        50 44FD   01B0          MULG2   R0, R2                  ; R2/R3 <-- 2^7*z*p(z*z)
                                01B4
                                01B4  ;
                                01B4  ; Compute log2(x) = k + i/16 + z*p(z)
                                01B4  ;
                                01B4
            50        54 4EFD   01B4          CVTLG   R4, R0                  ; Convert 2^7*(k + i/16) to double
            52        50 40FD   01B8          ADDG2   R0, R2                  ; R2/R3 <-- 2^7*log2(x)
                                01BC
                                01BC  ;
                                01BC  ; We can now compute x**y as 2^[y*log2(x)].  We begin by computing
                                01BC  ; y*log2(x).  (Note that R1 = 0.)
                                01BC  ;
                                01BC
       50   08 AC          50   01BC          MOVF    exp(AP), R0             ; R0/R1 <-- y
            50        50 99FD   01C0          CVTFG   R0, R0
                                01C4  ;
                                01C4  ; Test for the possibility of overflow in the computation of y*w1.
                                01C4  ; This will occur if the exponent of y plus the exponent of w1 is greater
```

```
                                01C4   409 ; than 127.
                                01C4   410 ;
      7E   50    0B    04    EF 01C4   411         EXTZV   #4, #11, R0, -(SP)      ; biased exp of y
           6E  0400    8F    A2 01C9   412         SUBW2   #^X400, (SP)            ; unbiased exp of y
      54   52    0B    04    EF 01CE   413         EXTZV   #4, #11, R2, R4         ; biased exp of 2^7*log2(x)
           54  0400    8F    A2 01D3   414         SUBW2   #^X400, R4              ; unbiased exp of 2^7*log2(x)
                 54    8E    C0 01D8   415         ADDL2   (SP)+, R4               ; unbiased exp of 2^7*log2(x)*y
           54  007F    8F    B1 01DB   416         CMPW    #^X7F, R4               ; largest unbiased exp possible is 127
                       03    18 01E0   417         BGEQ    NO_SYS_OVERFLOW
                     008C    31 01E2   418         BRW     Y_TIMES_W1_OVER
                                01E5   419 NO_SYS_OVERFLOW:
                 52    50  44FD 01E5   420         MULG2   `), R2                  ; R2/R3 <-- 2^7*y*log2(x)
                                01E9   421
                                01E9   422 ;
                                01E9   423 ; The next step in computing 2^[y*log2(x)] is to write y*log2(x) as
                                01E9   424 ;
                                01E9   425 ;               y*log2(x) = I + j/16 + g/16,
                                01E9   426 ;
                                01E9   427 ; where I is an integer, j is an integer between 0 and 15 inclusive, and
                                01E9   428 ; g is a fraction in the interval [-1/2, 1/2)
                                01E9   429 ;
                                01E9   430
 7E   53   FFFF1FFF    8F    CB 01E9   431         BICL3   #^XFFFF1FFF, R3, -(SP)
                 7E    52    D0 01F1   432         MOVL    R2, -(SP)
                 54    8E  33FD 01F4   433         CVTGF   (SP)+, R4
 50   54   00004DC0    8F    41 01F8   434         ADDF3   #^X4DC0, R4, R0         ; 3*2^5 is used in this truncation process
                                0200   435                                         ;  to avoid a possible normalization
                                0200   436                                         ;  that could occur if the number is neg
      50   00004DC0    8F    42 0200   437         SUBF    #^X4DC0, R0             ; R0/R1 <-- 2^7(I + j/16) in double
                 50    50  99FD 0207   438         CVTFG   R0, R0
                 52    50  42FD 020B   439         SUBG2   R0, R2                  ; R2/R3 <-- 2^7(g/16)
                 54    50  49FD 020F   440         CVTGW   R0, R4                  ; R4 <-- 2^7(I + j/16) in integer
                       52    1D 0213   441         BVS     EXCEPTION_1             ; Branch if |I| is too large
                                0215   442
                                0215   443 ;
                                0215   444 ; We can now compute
                                0215   445 ;
                                0215   446 ;       x**y = 2^[y*log2(x)] = 2^[I + j/16 + g/16]
                                0215   447 ;
                                0215   448 ;            = (2^I)*[A*(B+1)] = 2^I*[A + A*B], where
                                0215   449 ;
                                0215   450 ; A = 2^(j/16) is obtained from the A_TABLE and B = 2^(g/16) - 1 is obtained
                                0215   451 ; by a min/max approximation whose coefficients compensate to the factor of
                                0215   452 ; 2^7.
                                0215   453 ;
                                0215   454
      53   FFFF1FFF    8F    CA 0215   455         BICL2   #^XFFFF1FFF, R3
                 52    52  33FD 021C   456         CVTGF   R2, R2
      FEEA CF    03    52    55 0220   457         POLYF   R2, #EXPLEN-1, EXPTAB   ; R0 <-- B =  2^(g/16) - 1
 52   54   FFFFFF80    8F    CB 0226   458         BICL3   #^XFFFFFF80, R4, R2     ; R2 <-- index into A_TABLE table
      52    52    FD    8F    9C 022E  459         ROTL    #-3, R2, R2             ; R2 <-- index into A_TABLE table
        52  FE50 CF42    7D 0233       460         MOVQ    A_TABLE[R2], R2         ; R2/R3 <-- A = 2^(j/16)
 7E   53   FFFF1FFF    8F    CB 0239   461         BICL3   #^XFFFF1FFF, R3, -(SP)
                 7E    52    D0 0241   462         MOVL    R2, -(SP)
                 7E    8E  33FD 0244   463         CVTGF   (SP)+, -(SP)
                 50    8E    44 0248   464         MULF2   (SP)+, R0               ; R0 <-- A*B
                 50    50  99FD 024B   465         CVTFG   R0, R0
```

```
             50  52 40FD  024F           ADDG2   R2, R0                  ; R0/R1 <-- A + A*B
             50  50 33FD  0253           CVTGF   R0, R0                  ; R0 <--2^[(j + g)/16]
      54    007F 8F   AA  0257           BICW2   #^X7F, R4               ; R4 = 2^7*I
             50  54   A0  025C           ADDW2   R4, R0                  ; R0 <-- 2^I*2^[(j+g)/16]
   007F 8F       50   B1  025F           CMPW    R0, #^X7F               ; test for over/underflow
                 07   15  0264           BLEQ    EXCEPTION_2             ; see what exception is if neg or = 0
                      04  0266   RETURN: RET                             ; otherwise return result in R0
                          0267
                          0267   ;
                          0267   ; Handlers for software detected over/underflow conditions follow
                          0267   ;
                          0267
                          0267   EXCEPTION_1:
                 50   53  0267           TSTF    R0                      ; if big ARG > 0 goto overflow
                 1D   18  0269           BGEQ    OVER                    ;    handler, otherwise go to
                 08   11  026B           BRB     UNDER                   ;    underflow handler
                          026D   EXCEPTION_2:
                 54   B5  026D           TSTW    R4                      ; test sign of I; if I >= 0
                 17   18  026F           BGEQ    OVER                    ; go to overflow handler
                          0271
                          0271   ;
                          0271   ; y*w1 would have caused a hardware system floating overflow error. If y<0,
                          0271   ; then we should return a result of 0 since result = 2^(y*(w1+w2)). Note,
                          0271   ; y can not be zero.
                          0271   ;
                          0271
                          0271   Y_TIMES_W1_OVER:
                 50   53  0271           TSTF    R0                      ; if y < 0 no overflow is needed
                 13   14  0273           BGTR    OVER                    ; overflow for y > 0
                          0275
                          0275   ;
                          0275   ; Underflow; if user has FU set, signal error.  Always return 0.0
                          0275   ;
                          0275
                 50   D4  0275   UNDER:  CLRL    R0                      ; R0 = result.
    0B 04 AD     06   E1  0277           BBC     #6, SF$W_SAVE_PSW(FP), 2$
                          027C                                           ; has user enabled floating underflow?
      7E     00'8F    9A  027C           MOVZBL  #MTH$K_FLOUNDMAT, -(SP) ; trap code for hardware floating
                          0280                                           ; underflow.  Convert to MTH$_FLOUNDMAT
                          0280                                           ; (32-bit VAX-11 exception code)
  00000000'GF    01   FB  0280           CALLS   #1, G^MTH$$SIGNAL       ; signal condition
                      04  0287   2$:     RET                             ; return
                          0288
                          0288   ;
                          0288   ; Signal floating overflow, return reserved operand, -0.0
                          0288   ;
                          0288
      7E     00'8F    9A  0288   OVER:   MOVZBL  #MTH$K_FLOOVEMAT, -(SP) ; Move overflow code to stack
      50   01    0F   79  028C           ASHQ    #15, #1, R0             ; R0 = result = reserved operand -0.0.
                          0290                                           ; R0 will be copied to signal mechanism
                          0290                                           ; vector (CHF$L_MCH_R0/R1) so it can be
                          0290                                           ; fixed up by any error handler
  00000000'GF    01   FB  0290           CALLS   #1, G^MTH$$SIGNAL       ; signal condition
                      04  0297           RET                             ; return - R0 restored from CHF$L_MCH_R0/R1
                          0298
                          0298           .END
```

```
ACMASK           = 0000001C
A_TABLE            00000088 R      02
BASE             = 00000004
C0                 00000080 R      02
C2               = 384F43F6
C4               = C0234393
DEFINED            00000142 R      02
EVAL_LOG2          00000154 R      02
EXCEPTION_1        00000267 R      02
EXCEPTION_2        0000026D R      02
EXP              = 00000008
EXPLEN           = 00000004
EXPTAB             00000110 R      02
INDEX              00000000 R      02
MTH$$SIGNAL        ********    X   00
MTH$K_FLOOVEMAT    ********    X   00
MTH$K_FLOUNDMAT    ********    X   00
MTH$K_UNDEXP       ********    X   00
NO_SYS_OVERFLOW    000001E5 R      02
OTS$POWRR          00000120 RG     02
OVER               00000288 R      02
RETURN             00000266 R      02
SF$W_SAVE_PSW    = 00000004
UNDEFINED          00000131 R      02
UNDER              00000275 R      02
Y_TIMES_W1_OVER    00000271 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                       00000298  (  664.)  02 (  2.)    PIC    USR   CON   REL   LCL   SHR   EXE   RD  NOWRT NOVEC LONG

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

Phase                      Page faults   CPU Time       Elapsed Time
-----                      -----------   --------       ------------
Initialization                     35    00:00:00.10    00:00:00.88
Command processing                107    00:00:00.56    00:00:02.62
Pass 1                            128    00:00:01.87    00:00:05.91
Symbol table sort                   0    00:00:00.04    00:00:00.04
Pass 2                            104    00:00:01.15    00:00:05.10
Symbol table output                 4    00:00:00.04    00:00:00.21
Psect synopsis output               2    00:00:00.03    00:00:00.03
Cross-reference output              0    00:00:00.00    00:00:00.00
Assembler run totals              382    00:00:03.79    00:00:14.80

The working set limit was 1050 pages.
9361 bytes (19 pages) of virtual memory were used to buffer the intermediate code.
There were 10 pages of symbol table space allocated to hold 54 non-local and 1 local symbols.
521 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$:UVXPOWRR/OBJ=OBJ$:UVXPOWRR MSRC$:UVXPOWRR/UPDATE=(ENH$:UVXPOWRR)
```

OTSPOWHH
LIS

OTSPOWII
LIS

OTSPOWRJ
LIS

UVXPOWGG
LIS

OTSPOWGLU
LIS

OTSPOWHLU
LIS

OTSPOWHJ
LIS

UVXPOWCJ
LIS

UVXPOWRR
LIS

OTSPOWLUL
LIS

OTSPOWRR
LIS

OTSPOWJJ
LIS

UVXEXP
LIS

UVXGSINCO
LIS

OTSPOWRLU
LIS

0266 AH-BT13A-SE
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY

NCP
MAP

UVXSQRTR2
LIS

UVXSQRT
LIS

NCPDEF
SDL

UVXSINCOS
LIS

NMADEF
SDL

NCP