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

FILEID**OTSPOWRR

```
  000000    TTTTTTTTTT    SSSSSSSS  PPPPPPPP      000000    WW      WW  RRRRRRRR    RRRRRRRR
  000000    TTTTTTTTTT    SSSSSSSS  PPPPPPPP      000000    WW      WW  RRRRRRRR    RRRRRRRR
00      00      TT      SS          PP      PP  00      00  WW      WW  RR      RR  RR      RR
00      00      TT      SS          PP      PP  00      00  WW      WW  RR      RR  RR      RR
00      00      TT      SS          PP      PP  00      00  WW      WW  RR      RR  RR      RR
00      00      TT      SS          PP      PP  00      00  WW      WW  RR      RR  RR      RR
00      00      TT        SSSSSS    PPPPPPPP    00      00  WW      WW  RRRRRRRR    RRRRRRRR
00      00      TT        SSSSSS    PPPPPPPP    00      00  WW      WW  RRRRRRRR    RRRRRRRR
00      00      TT              SS  PP          00      00  WW  WW  WW  RR  RR      RR  RR
00      00      TT              SS  PP          00      00  WW  WW  WW  RR  RR      RR  RR
00      00      TT              SS  PP          00      00  WWWW  WWWW  RR    RR    RR    RR      ....
00      00      TT              SS  PP          00      00  WWWW  WWWW  RR    RR    RR    RR      ....
  000000        TT      SSSSSSSS    PP            000000    WW      WW  RR      RR  RR      RR    ....
  000000        TT      SSSSSSSS    PP            000000    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     1          .TITLE  OTS$POWRR - REAL ** REAL power routine
0000     2          .IDENT  /2-006/         ; File: OTSPOWRR.MAR  Edit: JCW2006
0000     3
0000     4  ;
0000     5  ;****************************************************************************
0000     6  ;*                                                                          *
0000     7  ;*  COPYRIGHT (c) 1978, 1980, 1982, 1984 BY                                 *
0000     8  ;*  DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.                  *
0000     9  ;*  ALL RIGHTS RESERVED.                                                    *
0000    10  ;*                                                                          *
0000    11  ;*  THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED    *
0000    12  ;*  ONLY IN  ACCORDANCE WITH  THE  TERMS  OF  SUCH  LICENSE  AND WITH THE   *
0000    13  ;*  INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR  ANY  OTHER    *
0000    14  ;*  COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY    *
0000    15  ;*  OTHER PERSON.  NO TITLE TO AND OWNERSHIP OF  THE  SOFTWARE IS  HEREBY    *
0000    16  ;*  TRANSFERRED.                                                            *
0000    17  ;*                                                                          *
0000    18  ;*  THE INFORMATION IN THIS SOFTWARE IS  SUBJECT TO CHANGE WITHOUT NOTICE    *
0000    19  ;*  AND  SHOULD  NOT  BE  CONSTRUED AS  A COMMITMENT BY DIGITAL EQUIPMENT    *
0000    20  ;*  CORPORATION.                                                            *
0000    21  ;*                                                                          *
0000    22  ;*  DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE  OR  RELIABILITY OF ITS    *
0000    23  ;*  SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.                 *
0000    24  ;*                                                                          *
0000    25  ;*                                                                          *
0000    26  ;****************************************************************************
0000    27  ;
0000    28  ;
0000    29  ; FACILITY: Language support library - user callable
0000    30  ;++
0000    31  ; ABSTRACT:
0000    32  ;
0000    33  ;       REAL base to REAL power.
0000    34  ;
0000    35  ;       Floating overflow can occur
0000    36  ;       Undefined exponentiation can occur if:
0000    37  ;               1)  Negative base
0000    38  ;               2)  0 base and power is 0 or negative.
0000    39  ;
0000    40  ;
0000    41  ;--
0000    42  ;
0000    43  ; VERSION: 2
0000    44  ;
0000    45  ; HISTORY:
0000    46  ;
0000    47  ; AUTHOR:
0000    48  ;       Bob Hanek, 3-Mar-83: Version 2
0000    49  ;
0000    50  ; MODIFIED BY:
0000    51  ;
0000    52  ;
0000    53  ;
```

```
0000     55          .SBTTL  HISTORY         ; Detailed current edit history
0000     56
0000     57
0000     58 ; Edit history for Version 2 of OTS$POWRR
0000     59 ;
0000     60 ; 2-001 New algorithm implemented. RNH 3-Mar-83
0000     61 ; 2-002 Make INDEX table a local table. LEB  24-May-1983
0000     62 ; 2-003 Change reference of INDEX(R2) to be INDEX[R2].  LEB 25-May-1983
0000     63 ; 2-004 Change reference of A_TABLE(Rx) to be A_TABLE[Rx].  LEB 26-May-1983
0000     64 ; 2-005 Added two ROTL #-3,Rx,Rx instructions to scale the value of Rx back
0000     65 ;       from 'index*2^3' to 'index' before A_TABLE[Rx] is referenced. The
0000     66 ;       INDEX was not scaled back to yield values of 'index' instead of
0000     67 ;       'index*2^3' because the mathematics of the code used does need the
0000     68 ;       value of index*2^3 in several computations. JCW 7-Jun-1983
0000     69 ; 2-006 Corrected a bug involving a SYS_F_FLTOVF_F error during a MULD RO, R2.
0000     70 ;       Code was added to see if a MTH overflow message or a zero should be
0000     71 ;       returned.  JCW 19-Jan-1984
0000     72 ;
0000     73 ;
```

```
                     0000     75          .SBTTL  DECLARATIONS
                     0000     76
                     0000     77 ;
                     0000     78 ; INCLUDE FILES:
                     0000     79 ;
                     0000     80
                     0000     81 ;
                     0000     82 ; EXTERNAL SYMBOLS:
                     0000     83 ;
                     0000     84          .DSABL  GBL
                     0000     85          .EXTRN  MTH$K_UNDEXP            ; Undefined exponentiation
                     0000     86          .EXTRN  MTH$K_FLOUNDMAT         ; Underflow
                     0000     87          .EXTRN  MTH$K_FLOOVEMAT         ; Overflow
                     0000     88          .EXTRN  MTH$$SIGNAL             ; Math error routine
                     0000     89 ;
                     0000     90 ; MACROS:
                     0000     91 ;
                     0000     92          $SFDEF                          ; Define stack frame symbols
                     0000     93
                     0000     94 ;
                     0000     95 ; EQUATED SYMBOLS:
                     0000     96 ;
00000004             0000     97          base    = 4                     ; base input formal - by-value
00000008             0000     98          exp     = 8                     ; exponent input formal - by-value
0000001C             0000     99          ACMASK  = ^M< R2, R3, R4>       ; register saving mask
384F43F6             0000    100          C2      = ^X384F43F6
C0234393             0000    101          C4      = ^XC0234393
                     0000    102 ;
                     0000    103 ; OWN STORAGE:
                     0000    104 ;
                     0000    105 ;       none
                     0000    106 ;
                     0000    107 ;
                     0000    108 ; PSECT DECLARATIONS:
                     0000    109 ;
                 00000000    110          .PSECT  _OTS$CODE       PIC,SHR,LONG,EXE,NOWRT
                     0000    111                                          ; program section for OTS$ code
                     0000    112 ;
                     0000    113 ; CONSTANTS:
                     0000    114 ;
                     0000    115
                     0000    116 ;
                     0000    117 ; The INDEX table gives the byte offset into the A_TABLE necessary to select
                     0000    118 ; the proper choice of 'a.'
                     0000    119 ;
08 08 08 08 08 00 00 00      0000    120 INDEX:   .BYTE   ^X00, ^X00, ^X00, ^X08, ^X08, ^X08, ^X08, ^X08
18 10 10 10 10 10 10 08      0008    121          .BYTE   ^X08, ^X10, ^X10, ^X10, ^X10, ^X10, ^X10, ^X18
20 20 20 18 18 18 18 18      0010    122          .BYTE   ^X18, ^X18, ^X18, ^X18, ^X18, ^X20, ^X20, ^X20
28 28 28 28 20 20 20 20      0018    123          .BYTE   ^X20, ^X20, ^X20, ^X20, ^X28, ^X28, ^X28, ^X28
30 30 30 30 30 30 28 28      0020    124          .BYTE   ^X28, ^X28, ^X30, ^X30, ^X30, ^X30, ^X30, ^X30
38 38 38 38 38 38 30 30      0028    125          .BYTE   ^X30, ^X30, ^X38, ^X38, ^X38, ^X38, ^X38, ^X38
40 40 40 40 40 40 40 38      0030    126          .BYTE   ^X38, ^X40, ^X40, ^X40, ^X40, ^X40, ^X40, ^X40
48 48 48 48 48 48 48 40      0038    127          .BYTE   ^X40, ^X48, ^X48, ^X48, ^X48, ^X48, ^X48, ^X48
50 50 50 50 50 50 50 48      0040    128          .BYTE   ^X48, ^X50, ^X50, ^X50, ^X50, ^X50, ^X50, ^X50
58 58 58 58 58 58 50 50      0048    129          .BYTE   ^X50, ^X50, ^X58, ^X58, ^X58, ^X58, ^X58, ^X58
60 60 60 60 60 58 58 58      0050    130          .BYTE   ^X58, ^X58, ^X58, ^X60, ^X60, ^X60, ^X60, ^X60
68 68 68 68 60 60 60 60      0058    131          .BYTE   ^X60, ^X60, ^X60, ^X60, ^X68, ^X68, ^X68, ^X68
```

```
70 70 68 68 68 68 68 68  0060   132          .BYTE   ^X68, ^X68, ^X68, ^X68, ^X68, ^X68, ^X70, ^X70
70 70 70 70 70 70 70 70  0068   133          .BYTE   ^X70, ^X70, ^X70, ^X70, ^X70, ^X70, ^X70, ^X70
78 78 78 78 78 78 78 78  0070   134          .BYTE   ^X78, ^X78, ^X78, ^X78, ^X78, ^X78, ^X78, ^X78
80 80 80 80 80 78 78 78  0078   135          .BYTE   ^X78, ^X78, ^X78, ^X80, ^X80, ^X80, ^X80, ^X80
                         0080   136
        1967295C AA3B44B8 0080   137 C0:      .QUAD   ^X1967295CAA3B44B8
                         0088   138
                         0088   139 ;
                         0088   140 ; The ith entry of the A_TABLE contains the value 2^(i/16)
                         0088   141 ;
                         0088   142
        00000000 00004080 0088   143 A_TABLE:.QUAD   ^X0000000000004080      ; 2^( 0/16)
        487B67CC AAC34085 0090   144          .QUAD   ^X487B67CCAAC34085      ; 2^( 1/16)
        8BD7E3EA 95C1408B 0098   145          .QUAD   ^X8BD7E3EA95C1408B      ; 2^( 2/16)
        11C373AB C3D34091 00A0   146          .QUAD   ^X11C373ABC3D34091      ; 2^( 3/16)
        B8A9518D 37F04098 00A8   147          .QUAD   ^XB8A9518D37F04098      ; 2^( 4/16)
        A1126091 F532409E 00B0   148          .QUAD   ^XA1126091F532409E      ; 2^( 5/16)
        5139A9B1 FED640A5 00B8   149          .QUAD   ^X5139A9B1FED640A5      ; 2^( 6/16)
        A14BEA42 583E40AD 00C0   150          .QUAD   ^XA14BEA42583E40AD      ; 2^( 7/16)
        DE6533F9 0[r340B5 00C8   151          .QUAD   ^XDE6533F904F340B5      ; 2^( 8/16)
        0C379F58 08A340BD 00D0   152          .QUAD   ^X0C379F5808A340BD      ; 2^( 9/16)
        06DB1155 672A40C5 00D8   153          .QUAD   ^X06DB1155672A40C5      ; 2^(10/16)
        8481151F 248C40CE 00E0   154          .QUAD   ^X8481151F248C40CE      ; 2^(11/16)
        9D6BCAD6 44FC40D7 00E8   155          .QUAD   ^X9D6BCAD644FC40D7      ; 2^(12/16)
        94E1EC2A CCDE40E0 00F0   156          .QUAD   ^X94E1EC2ACCDE40E0      ; 2^(13/16)
        2439E7DD C0C640EA 00F8   157          .QUAD   ^X2439E7DDC0C640EA      ; 2^(14/16)
        86CC1524 257D40F5 0100   158          .QUAD   ^X86CC1524257D40F5      ; 2^(15/16)
        00000000 00004100 0108   159          .QUAD   ^X0000000000004100      ; 2^(16/16)
                         0110   160
                59FC33E3 0110   161 EXPTAB: .LONG    ^X59FC33E3
                00663876 0114   162          .LONG   ^X00663876
                72183CB1 0118   163          .LONG   ^X72183CB1
                9625B19D 011C   164          .LONG   ^X9625B19D
                00000004 0120   165 EXPLEN = <.-EXPTAB>/4
```

```
0120   167          .SBTTL  OTS$POWRR - REAL to REAL giving REAL result
0120   168
0120   169 ;++
0120   170 ; FUNCTIONAL DESCRIPTION:
0120   171 ;
0120   172 ;       OTS$POWRR - REAL result = REAL base ** REAL exponent
0120   173 ;
0120   174 ;       The REAL result is given by:
0120   175 ;
0120   176 ;       base    exponent        result
0120   177 ;       ----    --------        ------
0120   178 ;
0120   179 ;       = 0     > 0             0.0
0120   180 ;       = 0     = 0             Undefined Exponentiation
0120   181 ;       = 0     < 0             Undefined Exponentiation
0120   182 ;
0120   183 ;       < 0     any             Undefined Exponentiation
0120   184 ;
0120   185 ;       > 0     > 0             2^[exp*log2(base)]
0120   186 ;       > 0     = 0             1.0
0120   187 ;       > 0     < 0             2^[exp*log2(base)]
0120   188 ;
0120   189 ;
0120   190 ;       Floating Overflow and Underflow can occur.
0120   191 ;       Undefined Exponentiation can occur if:
0120   192 ;               1)  base is 0 and exponent is 0 or negative
0120   193 ;               2)  base is negative
0120   194 ;
0120   195 ; The basic approach to computing x**y as 2^[y*log2(x)] is the following:
0120   196 ;
0120   197 ;       Step 1: Compute log2(x) to sufficient precision to guarantee an
0120   198 ;               accurate final result (see below.)
0120   199 ;       Step 2: Compute y*log2(x) to at least the accuracy that log2(x)
0120   200 ;               was computed.
0120   201 ;       Step 3: Evaluate 2^[y*log2(x)] accurate to the precision of the
0120   202 ;               datatype in question.
0120   203 ;
0120   204 ; To determine the accuracy to which log2(x) must be computed to, write
0120   205 ; y*log2(x) as I + h, where I is the integer closest to y*log2(x), and
0120   206 ; h = y*log2(x) - I (Note that |h| =< 1/2.)  Then
0120   207 ;
0120   208 ;               2^[y*log2(x)] = 2^(I + h) = (2^I)*(2^h).
0120   209 ;
0120   210 ; Since the factor 2^I can be incorporated into the final result by an integer
0120   211 ; addition to the exponent field, we can assume that the multiplication by
0120   212 ; 2^I incurs no error.  Thus the total error in the final result is determined
0120   213 ; by how accurately 2^h can be computed.  If the final result has p fraction
0120   214 ; bits, we would like h to have at least p good bits.  In fact it would be
0120   215 ; nice if h had a few extra guard bits, say 4.  Consequently, we would like
0120   216 ; h to be accurate to p + 4 bits.
0120   217 ;
0120   218 ; Let e be the number of bits allocated to the exponent field of the data type
0120   219 ; in question.  If I requires more that e bits to represent, then overflow or
0120   220 ; underflow will occur.  Therefore if the product y*log2(x) has e + p + 4 good
0120   221 ; bits, the final result will be accurate.  This requires that log2(x) be
0120   222 ; computed to at least p + e + 4 bits.
0120   223 ;
```

```
0120   224 ; Since log2(x) must be computed to more bits of precision than is available
0120   225 ; in the base data type, either the next level of precision or multi-precision
0120   226 ; arithmetic must be used.  We begin by writing
0120   227 ;
0120   228 ;
0120   229 ;                                    r---          2n+1
0120   230 ;           log2(x) = log2(b) +      >   c(2n+1)*z'
0120   231 ;                                    /
0120   232 ;                                    n=0
0120   233 ;
0120   234 ; Where c(1) = 1, and z' = (2/ln2)[(z-b)/(z+b)].  Hence
0120   235 ;
0120   236 ;
0120   237 ;                                    r---          2n+1
0120   238 ;           log2(x) = log2(b) + z' + >   c(2n+1)*z'
0120   239 ;                                    /
0120   240 ;                                    n=1
0120   241 ;
0120   242 ;                   = log2(b) + z' + p(z').
0120   243 ;
0120   244 ; Note that if p(z') is computed to p bits, and log2(b) + z' is computed
0120   245 ; to p+e+4 bits and overhangs p(z') by e+4 bits, the required accuracy will
0120   246 ; be achieved.  Consequently, the essential tricks, are to pick b such that
0120   247 ; the overhang can be achieved and to compute log2(b) + z' to p + e + 4 bits.
0120   248 ;
0120   249 ;
0120   250 ; CALLING SEQUENCE:
0120   251 ;
0120   252 ;       power.wf.v = OTS$POWRR (base.rf.v, exponent.rf.v)
0120   253 ;
0120   254 ; INPUT PARAMETERS:
0120   255 ;       Base and exponent parameters are call by value
0120   256 ;
0120   257 ; IMPLICIT INPUTS:
0120   258 ;       none
0120   259 ;
0120   260 ; OUTPUT PARAMETERS:
0120   261 ;       none
0120   262 ;
0120   263 ; IMPLICIT OUTPUTS:
0120   264 ;       none
0120   265 ;
0120   266 ; FUNCTIONAL VALUE:
0120   267 ;       OTS$POWRR - REAL base ** REAL power
0120   268 ;
0120   269 ; SIDE EFFECTS:
0120   270 ;
0120   271 ;       SIGNALs MTH$K_FLOOVEMAT if floating overflow.
0120   272 ;       SIGNALs MTH$K_FLOUNDMAT if floating underflow.
0120   273 ;       SIGNALs MTH$K_UNDEXP (82 = ' UNDEFINED EXPONENTIATION') if
0120   274 ;               1)  base is 0 and exponent is 0 or negative
0120   275 ;               2)  base is negative
0120   276 ;
0120   277 ;
0120   278 ;--
```

```
                 001C  0120   280          .ENTRY  OTS$POWRR, ACMASK       ; standard call-by-reference entry
                       0122   281                                          ; disable DV (and FU)
                       0122   282
                       0122   283 ;
                       0122   284 ; Move x to R0.  If x < 0, or x = 0 and y =< 0, return 'UNDEFINED
                       0122   285 ; EXPONENTIATION'' error condition, otherwise attempt to compute x**y
                       0122   286 ;
                       0122   287
          50  04 AC     50   0122   288          MOVF    base(AP), R0            ; R0 <-- x
                 1A     14   0126   289          BGTR    DEFINED                 ; If x > 0 attempt to compute x**y
                 07     19   0128   290          BLSS    UNDEFINED               ; Branch to error code for x < 0
          51  08 AC     50   012A   291          MOVF    exp(AP), R1             ; R1 <-- y (Note that x = 0)
                 01     15   012E   292          BLEQ    UNDEFINED               ; Branch to error condition if y =< 0
                       0130   293
                       0130   294 ;
                       0130   295 ; If processing continues here, this implies that x = 0 and y > 0.  Return
                       0130   296 ; with x**y = 0
                       0130   297 ;
                       0130   298
                 04    0130   299          RET                             ; Return
                       0131   300
                       0131   301 ;
                       0131   302 ; If processing continues here, this implies that an undefined exponentiation
                       0131   303 ; was attempted.  Signal error and return
                       0131   304 ;
                       0131   305
                       0131   306 UNDEFINED:
       50  8000 8F     3C   0131   307          MOVZWL  #^X8000, R0             ; R0 <-- Reserved operand
         7E  00'8F     9A   0136   308          MOVZBL  #MTH$K_UNDEXP, -(SP)    ; Put error code on stack
00000000'GF    01     FB   013A   309          CALLS   #1, G^MTH$$SIGNAL       ; Convert error number to 32 bit
                       0141   310                                          ;   condition code and signal error.
                       0141   311                                          ;   NOTE:  Second argument is not re-
                       0141   312                                          ;   quired since there is no JSB entry.
                 04    0141   313          RET                             ; Return
                       0142   314
                       0142   315 ;
                       0142   316 ; If processing continues here will attempt to compute x**y as 2^[y*log2(x)].
                       0142   317 ; We begin by determining an integer k and a real mumber f such that x = 2^k*f,
                       0142   318 ; and 1 =< f < 2.
                       0142   319 ;
                       0142   320
                       0142   321 DEFINED:
54     50  FFFF807F 8F   CB   0142   322          BICL3   #^XFFFF807F, R0, R4     ; R4 <-- 2^7*(biased exponent of x)
       54  00004080 8F   C2   014A   323          SUBL    #^X4080, R4             ; R4 <-- 2^7*k = 2^7*(exponent_of_x - 1)
           50    54   C2   0151   324          SUBL    R4, R0                  ; R0 <-- f = 2*(fraction field of x)
                       0154   325
                       0154   326 ;
                       0154   327 ; We are now ready to compute log2(x).  This computation is based on the
                       0154   328 ; following identity:
                       0154   329 ;
                       0154   330 ;                                        2  ,---   1                          f-a
                       0154   331 ;  log2(2^k*f) = k + log2(a) + ------ *  >    ---- z^(2j+1), where z = -----.
                       0154   332 ;                                   ln(2)/____ 2j+1                        f+a
                       0154   333 ;
                       0154   334 ; We begin by determining a as b^i, where b = 2^(1/16) and i is between 0
                       0154   335 ; and 16 inclusive.  Specifically i is chosen by table look-up so that
                       0154   336 ; the magnitude of z is minimized.  Since log2(a) = i/16, we may write
```

```
                                0154   337 ;
                                0154   338 ;                       log2(2^k*f) = k + i/16 + z*p(z^2).
                                0154   339 ;
                                0154   340 ; Note that in order to insure an accurate result, log2(2^k*f) must be computed
                                0154   341 ; accurately to 36 bits.  This will require some double precision arithmetic.
                                0154   342 ;
                                0154   343
                                0154   344 EVAL_LOG2:
                      51    00  D0  0154   345          MOVL    #0, R1                      ; R0/R1 <-- f
          52    50    FFFFFF80 8F  CB  0157   346          BICL3   #^XFFFFFF80, R0, R2         ; R2 <-- index to INDEX table
                52    FE9C CF42  90  015F   347          MOVB    INDEX[R2], R2               ; R2 <-- i*2^3
                      54    52   C0  0165   348          ADDL    R2, R4                      ; R4 <-- 2^7*(k + i/16)
                52    52    FD 8F  9C  0168   349          ROTL    #-3, R2, R2                 ; R2 <-- i
                                016D   350                                                 ; R2 will be multiplied by 2^3 by
                                016D   351                                                 ;  table references like the line below.
                                016D   352                                                 ;  The linker will cause an error if
                                016D   353                                                 ;  () are used instead of [] for these
                                016D   354                                                 ;  table references.
             52   50    FF16 CF42  63  016D   355          SUBD3   A_TABLE[R2], R0, R2         ; R2/R3 <-- f - a (NOTE: result is
                                0174   356                                                 ;    exact, i.e. no roundoff error)
                50    0080 8F    A0  0174   357          ADDW    #^X80, R0                   ; R0/R1 <-- 2*f
                      50    52   62  0179   358          SUBD    R2, R0                      ; R0/R1 <-- f + a
                      52    50   66  017C   359          DIVD    R0, R2                      ; R2/R3 <-- z
                                017F   360
                                017F   361 ;
                                017F   362 ; Compute 2^7*z*p(z^2) = z*(c0* + c2*z^2 + c4*z^4), where the c's are chosen
                                017F   363 ; to minimize the absolute error of the approximation
                                017F   364 ;
                                017F   365
             51   52    52    45  017F   366          MULF3   R2, R2, R1                  ; R1 <-- z^2
     50      51    C0234393 8F    45  0183   367          MULF3   #C4, R1, R0                 ; R0 <-- c4*z^2
     50      50    384F43F6 8F    40  018B   368          ADDF    #C2, R0                     ; R0 <-- c2 + c4*z^2
                      50    51   44  0192   369          MULF    R1, R0                      ; R0 <-- c2*z^2 + c4*z^4
                      51    00   D0  0195   370          MOVL    #0, R1                      ; R0/R1 <-- c2*z^2 + c4*z^4
                50    FEE4 CF    60  0198   371          ADDD    C0, R0                      ; R0/R1 <-- c0 + c2*z^2 + c4*z^4
                      52    50   64  019D   372          MULD    R0, R2                      ; R2/R3 <-- 2^7*z*p(z*z)
                                01A0   373
                                01A0   374 ;
                                01A0   375 ; Compute log2(x) = k + i/16 + z*p(z)
                                01A0   376 ;
                                01A0   377
                      50    54   4E  01A0   378          CVTLF   R4, R0                      ; Convert 2^7*(k + i/16) to double
                      51    00   D0  01A3   379          MOVL    #0, R1                      ;
                      52    50   60  01A6   380          ADDD    R0, R2                      ; R2/R3 <-- 2^7*log2(x)
                                01A9   381
                                01A9   382 ;
                                01A9   383 ; We can now compute x**y as 2^[y*log2(x)].  We begin by computing
                                01A9   384 ; y*log2(x).  (Note that R1 = 0.)
                                01A9   385 ;
                                01A9   386
                50    08 AC    50  01A9   387          MOVF    exp(AP), R0                 ; R0/R1 <-- y
                                01AD   388 ;
                                01AD   389 ; Test for the possibility of overflow in the computation of y*w1.
                                01AD   390 ; This will occur if the exponent of y plus the exponent of w1 is greater
                                01AD   391 ; than 127.
                                01AD   392 ;
                                01AD   393
```

```
                7E    50    08    07    EF   01AD   394          EXTZV   #7, #8, R0, -(SP)       ; biased exp of y
                   6E 0080    8F    A2   01B2   395          SUBW2   #^X80, (SP)              ; unbiased exp of y
                54  52    08    07    EF   01B7   396          EXTZV   #7, #8, R2, R4          ; biased exp of 2^7*log2(x)
                    54 0080    8F    A2   01BC   397          SUBW2   #^X80, R4               ; unbiased exp of 2^7*log2(x)
                       54      8E    C0   01C1   398          ADDL2   (SP)+, R4               ; unbiased exp of 2^7*log2(x)*y
                    54 007F    8F    B1   01C4   399          CMPW    #^X7F, R4               ; largest unbiased exp possible is 127
                              56    19   01C9   400          BLSS    Y_TIMES_W1_OVER
                          52  50    64   01CB   401          MULD    R0, R2                  ; R2/R3 <-- 2^7*y*log2(x)
                                    01CE   402
                                    01CE   403 ;
                                    01CE   404 ; The next step in computing 2^[y*log2(x)] is to write y*log2(x) as
                                    01CE   405 ;
                                    01CE   406 ;                   y*log2(x) = I + j/16 + g/16,
                                    01CE   407 ;
                                    01CE   408 ; where I is an integer, j is an integer between 0 and 15 inclusive, and
                                    01CE   409 ; g is a fraction in the interval [-1/2, 1/2)
                                    01CE   410 ;
                                    01CE   411
          50    52  00004DC0 8F     41   01CE   412          ADDF3   #^X4DC0, R2, R0         ; 3*2^5 is used in this truncation process
                                    01D6   413                                              ;  to avoid a possible normalization
                                    01D6   414                                              ;  that could occur if the number is neg
                50  00004DC0 8F     42   01D6   415          SUBF    #^X4DC0, R0             ; R0/R1 <-- 2^7(I + j/16) in double
                          52  50    62   01DD   416          SUBD    R0, R2                  ; R2/R3 <-- 2^7(g/16)
                          54  50    49   01E0   417          CVTFW   R0, R4                  ; R4 <-- 2^7(I + j/16) in integer
                              32    1D   01E3   418          BVS     EXCEPTION_1             ; Branch if !I! is too large
                                    01E5   419
                                    01E5   420 ;
                                    01E5   421 ; We can now compute
                                    01E5   422 ;
                                    01E5   423 ;          x**y = 2^[y*log2(x)] = 2^[I + j/16 + g/16]
                                    01E5   424 ;
                                    01E5   425 ;               = (2^I)*[A*(B+1)] = 2^I*[A + A*B], where
                                    01E5   426 ;
                                    01E5   427 ; A = 2^(j/16) is obtained from the A_TABLE and B = 2^(g/16) - 1 is obtained
                                    01E5   428 ; by a min/max approximation whose coefficients compensate to the factor of
                                    01E5   429 ; 2^7.
                                    01E5   430 ;
                                    01E5   431
       FF25 CF    03    52    55   01E5   432          POLYF   R2, #EXPLEN-1, EXPTAB   ; R0 <-- B =  2^(g/16) - 1
52      54   FFFFFF80 8F    CB   01EB   433          BICL3   #^XFFFFFF80, R4, R2     ; R2 <-- 2^3 * index into A_TABLE table
          52    52    FD 8F    9C   01F3   434          ROTL    #-3, R2, R2             ; R2 <-- index into A_TABLE table
           52   FE8B CF42    7D   01F8   435          MOVQ    A_TABLE[R2], R2         ; R2/R3 <-- A = 2^(j/16)
                     50    52   44   01FE   436          MULF    R2, R0                  ; R0 <-- A*B
                     50    52   60   0201   437          ADDD    R2, R0                  ; R0/R1 <-- A + A*B
                     50    50   76   0204   438          CVTDF   R0, R0                  ; R0 <--2^[(j + g)/16]
                54   007F 8F    AA   0207   439          BICW    #^X7F, R4               ; R4 = 2^7*I
                     50    54   A0   020C   440          ADDW    R4, R0                  ; R0 <-- 2^I*2^[(j+g)/16]
                007F 8F    50   B1   020F   441          CMPW    R0, #^X7F               ; test for over/underflow
                           07   15   0214   442          BLEQ    EXCEPTION_2             ; see what exception is if neg or = 0
                           04   0216   443 RETURN: RET                                   ; otherwise return result in R0
                                    0217   444
                                    0217   445 ;
                                    0217   446 ; Handlers for software detected over/underflow conditions follow
                                    0217   447 ;
                                    0217   448
                                    0217   449 EXCEPTION_1:
                          50    53   0217   450          TSTF    R0                      ; if big ARG > 0 goto overflow
```

```
           1D   18  0219          BGEQ    OVER                    ;    handler, otherwise go to
           08   11  021B          BRB     UNDER                   ;    underflow handler
                    021D  EXCEPTION_2:
           54   B5  021D          TSTW    R4                      ; test sign of I; if I >= 0
           17   18  021F          BGEQ    OVER                    ; go to overflow handler
                    0221
                    0221  ;
                    0221  ; y*w1 would have caused a hardware system floating overflow error. If y<0,
                    0221  ; then we should return a result of 0 since result = 2^(y*(w1+w2)). Note,
                    0221  ; y can not be zero.
                    0221  ;
                    0221
                    0221  Y_TIMES_W1_OVER:
           50   53  0221          TSTF    R0                      ; if y < 0 no overflow is needed
           13   14  0223          BGTR    OVER                    ; overflow
                    0225
                    0225
                    0225  ;
                    0225  ; Underflow; if user has FU set, signal error.  Always return 0.0
                    0225  ;
                    0225
           50   D4  0225  UNDER:  CLRL    R0                      ; R0 = result.
  0B 04 AD   06   E1  0227          BBC     #6, SF$W_SAVE_PSW(FP), 2$
                    022C                                          ; has user enabled floating underflow?
     7E   00'8F   9A  022C          MOVZBL  #MTH$K_FLOUNDMAT, -(SP) ; trap code for hardware floating
                    0230                                          ; underflow.  Convert to MTH$_FLOUNDMAT
                    0230                                          ; (32-bit VAX-11 exception code)
00000000'GF   01  FB  0230          CALLS   #1, G^MTH$$SIGNAL       ; signal condition
                04  0237  2$:     RET                             ; return
                    0238
                    0238  ;
                    0238  ; Signal floating overflow, return reserved operand, -0.0
                    0238  ;
                    0238
     7E   00'8F   9A  0238  OVER:   MOVZBL  #MTH$K_FLOOVEMAT, -(SP) ; Move overflow code to stack
   50   01   0F   79  023C          ASHQ    #15, #1, R0             ; R0 = result = reserved operand -0.0.
                    0240                                          ; R0 will be copied to signal mechanism
                    0240                                          ; vector (CHF$L_MCH_R0/R1) so it can be
                    0240                                          ; fixed up by any error handler
00000000'GF   01  FB  0240          CALLS   #1, G^MTH$$SIGNAL       ; signal condition
                04  0247          RET                             ; return - R0 restored from CHF$L_MCH_R0/R1
                    0248
                    0248          .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       00000217 R      02
EXCEPTION_2       0000021D 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$K_UNDEXP      ********    X   00
OTS$POWRR         00000120 RG     02
OVER              00000238 R      02
RETURN            00000216 R      02
SF$W_SAVE_PSW   = 00000004
UNDEFINED         00000131 R      02
UNDER             00000225 R      02
Y_TIMES_W1_OVER   00000221 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                       00000248  (  584.)  02 (  2.)    PIC   USR   CON   REL   LCL   SHR   EXE   RD  NOWRT NOVEC LONG
```

Performance indicators

Phase	Page faults	CPU Time	Elapsed Time
Initialization	31	00:00:00.07	00:00:00.81
Command processing	113	00:00:00.66	00:00:04.49
Pass 1	124	00:00:01.88	00:00:06.64
Symbol table sort	0	00:00:00.04	00:00:00.04
Pass 2	97	00:00:01.17	00:00:04.40
Symbol table output	3	00:00:00.03	00:00:00.03
Psect synopsis output	3	00:00:00.02	00:00:00.02
Cross-reference output	0	00:00:00.00	00:00:00.00
Assembler run totals	373	00:00:03.89	00:00:16.45

The working set limit was 900 pages.
9218 bytes (19 pages) of virtual memory were used to buffer the intermediate code.
There were 10 pages of symbol table space allocated to hold 53 non-local and 1 local symbols.
553 source lines were read in Pass 1, producing 13 object records in Pass 2.
9 pages of virtual memory were used to define 8 macros.

K 9

Macro library statistics

Macro library name	Macros defined
_255DUA28:[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$:OTSPOWRR/OBJ=OBJ$:OTSPOWRR MSRC$:MTHJACKET/UPDATE=(ENH$:MTHJACKET)+MSRC
```

0265 AH-BT13A-SE
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY

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