```
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
```

```
UU      UU  VV      VV  XX      XX  PPPPPPPP      000000    WW      WW    GGGGGGGG    GGGGGGGG
UU      UU  VV      VV  XX      XX  PPPPPPPP      000000    WW      WW    GGGGGGGG    GGGGGGGG
UU      UU  VV      VV  XX      XX  PP      PP  00      00  WW      WW  GG          GG
UU      UU  VV      VV  XX      XX  PP      PP  00      00  WW      WW  GG          GG
UU      UU  VV      VV    XX  XX    PP      PP  00      00  WW      WW  GG          GG
UU      UU  VV      VV    XX  XX    PP      PP  00      00  WW      WW  GG          GG
UU      UU  VV      VV      XX      PPPPPPPP    00      00  WW      WW  GG          GG
UU      UU  VV      VV      XX      PPPPPPPP    00      00  WW      WW  GG          GG
UU      UU  VV      VV    XX  XX    PP          00      00  WW  WW  WW  GG  GGGGGG  GG  GGGGGG
UU      UU  VV      VV    XX  XX    PP          00      00  WW  WW  WW  GG  GGGGGG  GG  GGGGGG
UU      UU    VV  VV    XX      XX  PP          00      00  WWWW  WWWW  GG      GG  GG      GG      ....
UU      UU    VV  VV    XX      XX  PP          00      00  WWWW  WWWW  GG      GG  GG      GG      ....
UUUUUUUUUU      VV      XX      XX  PP            000000    WW      WW    GGGGGG      GGGGGG        ....
UUUUUUUUUU      VV      XX      XX  PP            000000    WW      WW    GGGGGG      GGGGGG        ....

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  UVX$POWGG - G REAL*8 ** G REAL*8 power routine
0000     2          .IDENT  /2-006/                 ; File: OTSPOWGG.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 ;         G double base to G double 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 ;
0000    44 ; VERSION: 2
0000    45 ;
0000    46 ; HISTORY:
0000    47 ;
0000    48 ; AUTHOR:
0000    49 ;       Bob Hanek 8-Mar-83: Version 2
0000    50 ;
0000    51 ; MODIFIED BY:
0000    52 ;
0000    53 ;       Jeffrey C. Wiener 16-MAY-83: Version 2-002
0000    54 ;
```

```
                     0000     56          .SBTTL  HISTORY         ; Detailed current edit history
                     0000     57
                     0000     58
                     0000     59 ; Edit history for Version 2 of OTS$POWGG
                     0000     60 ;
                     0000     61 ; 2-001 - Implemented new algorithm. RNH 8-Mar-83
                     0000     62 ; 2-002 - MicroVAX requires that G_floating routines may not be backed-up
                     0000     63 ;         by H_floating point instructions. To accomplish this the code
                     0000     64 ;         from OTS$POWDD was modified to create this present version of
                     0000     65 ;         UVXPOWGG. All constants had to be scaled accordingly when taken
                     0000     66 ;         from the original OTS$POWGG to accomodate the difference between
                     0000     67 ;         z and z'=2^67ln(2). JCW 16-MAY-83.
                     0000     68 ; 2-003 - Change table INDEX to be local instead of global.  LEB 24-May-1983
                     0000     69 ; 2-004 - Change reference of INDEX(Rx) to be INDEX[Rx] to avoid linker
                     0000     70 ;         errors.  LEB 25-May-1983
                     0000     71 ; 2-005 - Changed MTH$POWGG entry to OTS$POWGG entry. JCW 26-May-1983
                     0000     72 ; 2-006 Corrected a bug involving a SYS_F_FLTOVF_F error during a MULG R6, R2.
                     0000     73 ;       Code was added to see if a MTH overflow message or a zero should be
                     0000     74 ;       returned.  JCW 19-Jan-1984
                     0000     75 ;
                     0000     76
                     0000     77          .SBTTL  DECLARATIONS
                     0000     78
                     0000     79 ;
                     0000     80 ; INCLUDE FILES:
                     0000     81 ;
                     0000     82
                     0000     83 ;
                     0000     84 ; EXTERNAL SYMBOLS:
                     0000     85 ;
                     0000     86
                     0000     87           DSABL  GBL
                     0000     88          .EXTRN  MTH$$SIGNAL             ; Error signal routine
                     0000     89          .EXTRN  MTH$K_UNDEXP            ; Undefined exponentiation code
                     000C     90          .EXTRN  MTH$K_FLOUNDMAT         ; Floating point underflow code
                     000C     91          .EXTRN  MTH$K_FLOOVEMAT         ; Floating point overflow code
                     0000     92
                     0000     93 ;
                     0000     94 ; MACROS:
                     0000     95 ;
                     0000     96          $SFDEF                          ; Define stack frame symbols
                     0000     97 ;
                     0000     98 ; EQUATED SYMBOLS:
                     0000     99 ;
00000004             0000    100          base    = 4                     ; base input formal - by-value
0000000C             0000    101          exp     = 12                    ; exponent input formal - by-value
00000008             0000    102          fexp    = 8                     ; exponent when base is floating
                     0000    103                                          ; by-value
000047FC             0000    104          ACMASK  = ^M<IV, R2, R3, R4, R5, R6, R7, R8, R9, R10>
                     0000    105
                     0000    106 ;
                     0000    107 ; OWN STORAGE: None
                     0000    108 ;
                     0000    109
                     0000    110 ;
                     0000    111 ; PSECT DECLARATIONS:
                     0000    112 ;
```

```
                                00000000   113          .PSECT  _OTS$CODE       PIC,SHR,LONG,EXE,NOWRT
                                    0000   114                                  ; program section for OTS$ code
                                    0000   115 ;
                                    0000   116 ; Constants
                                    0000   117 ;
                                    0000   118
                                    0000   119
                00000000 00003CA0   0000   120 TWO_M52:.QUAD   ^X0000000000003CA0
                                    0008   121
                                    0008   122 ;
                                    0008   123 ; The INDEX table gives the appropriate entry in the A_TABLE
                                    0008   124 ;
                                    0008   125
                                    0008   126 INDEX:
03 02 02 02 01 01 01 00             0008   127          .BYTE   ^X00, ^X01, ^X01, ^X01, ^X02, ^X02, ^X02, ^X03
05 05 05 04 04 04 03 03             0010   128          .BYTE   ^X03, ^X03, ^X04, ^X04, ^X04, ^X05, ^X05, ^X05
08 07 07 07 07 06 06 06             0018   129          .BYTE   ^X06, ^X06, ^X06, ^X07, ^X07, ^X07, ^X07, ^X08
0A 0A 0A 09 09 09 08 08             0020   130          .BYTE   ^X08, ^X08, ^X09, ^X09, ^X09, ^X0A, ^X0A, ^X0A
0C 0C 0C 0C 0B 0B 0B 0A             0028   131          .BYTE   ^X0A, ^X0B, ^X0B, ^X0B, ^X0C, ^X0C, ^X0C, ^X0C
0F 0E 0E 0E 0E 0D 0D 0D             0030   132          .BYTE   ^X0D, ^X0D, ^X0D, ^X0E, ^X0E, ^X0E, ^X0E, ^X0F
11 10 10 10 10 0F 0F 0F             0038   133          .BYTE   ^X0F, ^X0F, ^X0F, ^X10, ^X10, ^X10, ^X10, ^X11
13 12 12 12 12 11 11 11             0040   134          .BYTE   ^X11, ^X11, ^X11, ^X12, ^X12, ^X12, ^X12, ^X13
14 14 14 14 14 13 13 13             0048   135          .BYTE   ^X13, ^X13, ^X13, ^X14, ^X14, ^X14, ^X14, ^X14
16 16 16 16 15 15 15 15             0050   136          .BYTE   ^X15, ^X15, ^X15, ^X15, ^X16, ^X16, ^X16, ^X16
18 18 18 17 17 17 17 17             0058   137          .BYTE   ^X17, ^X17, ^X17, ^X17, ^X17, ^X18, ^X18, ^X18
1A 1A 19 19 19 19 18 18             0060   138          .BYTE   ^X18, ^X18, ^X19, ^X19, ^X19, ^X19, ^X1A, ^X1A
1B 1B 1B 1B 1B 1A 1A 1A             0068   139          .BYTE   ^X1A, ^X1A, ^X1A, ^X1B, ^X1B, ^X1B, ^X1B, ^X1B
1D 1D 1D 1C 1C 1C 1C 1C             0070   140          .BYTE   ^X1C, ^X1C, ^X1C, ^X1C, ^X1C, ^X1D, ^X1D, ^X1D
1E 1E 1E 1E 1E 1D 1D 1D             0078   141          .BYTE   ^X1D, ^X1D, ^X1D, ^X1E, ^X1E, ^X1E, ^X1E, ^X1E
20 20 20 1F 1F 1F 1F 1F             0080   142          .BYTE   ^X1F, ^X1F, ^X1F, ^X1F, ^X1F, ^X20, ^X20, ^X20
                                    0088   143
                                    0088   144 ;
                                    0088   145 ; For k = 0, 2, ..., 32, the k-th entry in the A_TABLE is the value 2^(k/32)
                                    0088   146 ; rounded to 113 bits.
                                    0088   147 ;
                                    0088   148
                                    0088   149 A1_TABLE:
        00000000 00004010           0088   150          .QUAD   ^X0000000000004010
        8574D315 59B04010           0090   151          .QUAD   ^X8574D31559B04010
        890F6CF9 B5584010           0098   152          .QUAD   ^X890F6CF9B5584010
        5B51D012 13014011           00A0   153          .QUAD   ^X5B51D01213014011
        517B3C7D 72B84011           00A8   154          .QUAD   ^X517B3C7D72B84011
        B9AA3168 D4874011           00B0   155          .QUAD   ^XB9AA3168D4874011
        62386E75 387A4012           00B8   156          .QUAD   ^X62386E75387A4012
        DEE1F51F 9E9D4012           00C0   157          .QUAD   ^XDEE1F51F9E9D4012
        B7150A31 06FE4013           00C8   158          .QUAD   ^XB7150A3106FE4013
        A9CB373A 71A74013           00D0   159          .QUAD   ^XA9CB373A71A74013
        34224C12 DEA64013           00D8   160          .QUAD   ^X34224C12DEA64013
        892D6061 4E084014           00E0   161          .QUAD   ^X892D60614E084014
        2A27D536 BFDA4014           00E8   162          .QUAD   ^X2A27D536BFDA4014
        4F82569D 342B4015           00F0   163          .QUAD   ^X4F82569D342B4015
        5429DD48 AB074015           00F8   164          .QUAD   ^X5429DD48AB074015
        5585B03A 247E4016           0100   165          .QUAD   ^X5585B03A247E4016
        3BCD667F A09E4016           0108   166          .QUAD   ^X3BCD667FA09E4016
        5F74E8EC 1F754017           0110   167          .QUAD   ^X5F74E8EC1F754017
        018773EB A1144017           0118   168          .QUAD   ^X018773EBA1144017
        CE13994C 25894018           0120   169          .QUAD   ^XCE13994C25894018
```

```
                                                I 15
A0DB422A ACE54018  0128   170          .QUAD   ^XA0DB422AACE54018
C5E5B0CD 37374019  0130   171          .QUAD   ^XC5E5B0CD37374019
F09082A3 C4914019  0138   172          .QUAD   ^XF09082A3C4914019
255DB23E 5503401A  0140   173          .QUAD   ^X255DB23E5503401A
D3AD995A E89F401A  0148   174          .QUAD   ^XD3AD995AE89F401A
5E47F2FB 7F76401B  0150   175          .QUAD   ^X5E47F2FB7F76401B
529CDD85 199B401C  0158   176          .QUAD   ^X529CDD85199B401C
9069DCEF B720401C  0160   177          .QUAD   ^X9069DCEFB720401C
A487DCFB 5818401D  0168   178          .QUAD   ^XA487DCFB5818401D
9B5F337B FC97401D  0170   179          .QUAD   ^X9B5F337BFC97401D
90DAA2A4 A4AF401E  0178   180          .QUAD   ^X90DAA2A4A4AF401E
45405B6E 5076401F  0180   181          .QUAD   ^X45405B6E5076401F
00000000 00004020  0188   182          .QUAD   ^X0000000000004020
                   0190   183
                   0190   184 A2_TABLE:
00000000 00000000  0190   185          .QUAD   ^X0000000000000000
B465A475 73E23CAD  0198   186          .QUAD   ^XB465A47573E23CAD
610B4ADC A62E3CB8  01A0   187          .QUAD   ^X610B4ADCA62E3CB8
9B3A3944 C510BCB6  01A8   188          .QUAD   ^X9B3A3944C510BCB6
8A76B9D7 9041BCA1  01B0   189          .QUAD   ^X8A76B9D79041BCA1
643C00A2 016E3CBE  01B8   190          .QUAD   ^X643C00A2016E3CBE
0573B6C7 B07E3CB9  01C0   191          .QUAD   ^X0573B6C7B07E3CB9
1255AFAD 12E83CA6  01C8   192          .QUAD   ^X1255AFAD12E83CA6
82E4D231 F46A3CA6  01D0   193          .QUAD   ^X82E4D231F46A3CA6
EAE2BF42 3AEABCB6  01D8   194          .QUAD   ^XEAE2BF423AEABCB6
9EBC11F0 DA093CAA  01E0   195          .QUAD   ^X9EBC11F0DA093CAA
80D004EF 9B7A3C68  01E8   196          .QUAD   ^X80D004EF9B7A3C68
42E2AFEC 43973C9D  01F0   197          .QUAD   ^X42E2AFEC43973C9D
3CAD1DB1 7ABEBCA0  01F8   198          .QUAD   ^X3CAD1DB17ABEBCA0
47AD0546 324C3CB6  0200   199          .QUAD   ^X47AD0546324C3CB6
B4977E40 83C1BCB3  0208   200          .QUAD   ^XB4977E4083C1BCB3
645613B2 DD34BCBB  0210   201          .QUAD   ^X645613B2DD34BCBB
7A998688 6E47BCA1  0218   202          .QUAD   ^X7A9986886E47BCA1
992FEE04 1577BCA4  0220   203          .QUAD   ^X992FEE041577BCA4
32D8D415 4C1DBCBD  0228   204          .QUAD   ^X32D8D4154C1DBCBD
4B275686 E9F13CB6  0230   205          .QUAD   ^X4B275686E9F13CB6
7EBC81B5 5FC7BC87  0238   206          .QUAD   ^X7EBC81B55FC7BC87
F2BEB071 7C463C9C  0240   207          .QUAD   ^XF2BEB0717C463C9C
41E1DB8D 2F6EBCBD  0248   208          .QUAD   ^X41E1DB8D2F6EBCBD
CC81345D A1CD3CB7  0250   209          .QUAD   ^XCC81345DA1CD3CB7
AC3B7E54 584FBC95  0258   210          .QUAD   ^XAC3B7E54584FBC95
48DD8950 10653CA1  0260   211          .QUAD   ^X48DD895010653CA1
49DBD1E9 03CB3C95  0268   212          .QUAD   ^X49DBD1E903CB3C95
3707D75B ED023CA2  0270   213          .QUAD   ^X3707D75BED023CA2
4B5C4F18 A5CDBCB1  0278   214          .QUAD   ^X4B5C4F18A5CDBCB1
2893179C 9C23BCBE  0280   215          .QUAD   ^X2893179C9C23BCBE
A18B2DD8 D3E13CB9  0288   216          .QUAD   ^XA18B2DD8D3E13CB9
00000000 00000000  0290   217          .QUAD   ^X0000000000000000
                   0298   218
                   0298   219
29DE98EE 39803D70  0298   220 LOGTAB:  .QUAD   ^X29DE98EE39803D70
BDB41F6F A3343E47  02A0   221          .QUAD   ^XBDB41F6FA3343E47
83C1FFAC 7FD33F24  02A8   222          .QUAD   ^X83C1FFAC7FD33F24
00000000 00000000  02B0   223          .QUAD   ^X0000000000000000
         00000004  02B8   224 LOGLEN = <.-LOGTAB>/8
                   02B8   225
82FE652B 15474077  02B8   226 C:       .QUAD   ^X82FE652B15474077
```

```
00006000 15474077  02C0    227 C1:     .QUAD   ^X0000600015474077
AE0BF85D AE0B3ED4  02C8    228 C2:     .QUAD   ^XAE0BF85DAE0B3ED4
                   02D0    229
                   02D0    230
FF06BA87 30963D64  02D0    231 EXPTAB: .QUAD   ^XFF06BA8730963D64      ;   0.14345715681320842E-12
1AA0E6EF D6853DE5  02D8    232         .QUAD   ^X1AA0E6EFD6853DE5      ;   0.39737266313176133E-10
7BD56FB9 B2AB3E63  02E0    233         .QUAD   ^X7BD56FB9B2AB3E63      ;   0.91725627017354366E-08
80DDD703 6B083EDC  02E8    234         .QUAD   ^X80DDD7036B083EDC      ;   0.16938509724215757E-05
C58FFF82 BFBD3F4E  02F0    235         .QUAD   ^XC58FFF82BFBD3F4E      ;   0.23459619820224680E-03
39EFFEFA 2E423FB6  02F8    236         .QUAD   ^X39EFFEFA2E423FB6      ;   0.21660849392498290E-01
00000000 00000000  0300    237         .QUAD   ^X0000000000000000      ;   0.00000000000000000E+00
         00000007  0308    238 EXPLEN = <.-EXPTAB>/8
                   0308    239
```

```
0308   241          .SBTTL  OTS$POWGG - G REAL*8 to G REAL*8 giving G REAL*8 result
0308   242
0308   243 ;++
0308   244 ; FUNCTIONAL DESCRIPTION:
0308   245 ;
0308   246 ;       OTS$POWGG - G REAL*8 result = G REAL*8 base ** G REAL*8 exponent
0308   247 ;
0308   248 ;       The G REAL*8 result is given by:
0308   249 ;
0308   250 ;       base    exponent        result
0308   251 ;       ----    --------        ------
0308   252 ;
0308   253 ;       = 0     > 0             0.0
0308   254 ;       = 0     = 0             Undefined Exponentiation
0308   255 ;       = 0     < 0             Undefined Exponentiation
0308   256 ;
0308   257 ;       < 0     any             Undefined Exponentiation
0308   258 ;
0308   259 ;       > 0     > 0             2^(exp * Log2(base))
0308   260 ;       > 0     = 0             1.0
0308   261 ;       > 0     < 0             2^(exp * Log2(base))
0308   262 ;
0308   263 ;
0308   264 ;       Floating Overflow can occur.
0308   265 ;       Undefined Exponentiation can occur if:
0308   266 ;               1)  base is 0 and exponent is 0 or negative
0308   267 ;               2)  base is negative
0308   268 ;
0308   269 ; CALLING SEQUENCE:
0308   270 ;
0308   271 ;       power.wg.v = OTS$POWGG (base.rg.v, exponent.rg.v)
0308   272 ;
0308   273 ; INPUT PARAMETERS:
0308   274 ;       base and exponent parameters are call by value
0308   275 ;
0308   276 ; IMPLICIT INPUTS:
0308   277 ;       none
0308   278 ;
0308   279 ; OUTPUT PARAMETERS:
0308   280 ;       none
0308   281 ;
0308   282 ; IMPLICIT OUTPUTS:
0308   283 ;       none
0308   284 ;
0308   285 ; FUNCTIONAL VALUE:
0308   286 ;       OTS$POWGG - G REAL*8 base ** G REAL*8 power
0308   287 ;
0308   288 ; SIDE EFFECTS:
0308   289 ;
0308   290 ;       SIGNALs floating overflow
0308   291 ;       SIGNALs floating underflow if underflow detection is enabled
0308   292 ;       SIGNALs MTH$_UNDEXP (82 = ' UNDEFINED EXPONENTIATION') if
0308   293 ;               1)  base is 0 and exponent is 0 or negative
0308   294 ;               2)  base is negative
0308   295 ;
0308   296 ;
0308   297 ;--
```

```
                          0308   299
                   47FC   0308   300          .ENTRY  OTS$POWGG, ACMASK       ;
                          030A   301 ;
                          030A   302 ; Move x to R0/R1.  If x < 0, or x = 0 and y =< 0, return "UNDEFINED
                          030A   303 ; EXPONENTIATION" error condition, otherwise attempt to compute x**y
                          030A   304 ;
                          030A   305
                          030A   306 GET_BASE:
          50    04 AC 50FD   030A   307          MOVG    base(AP), R0            ; R0/R1 <-- x
                          030F   308 COMMON:
                1C    14    030F   309          BGTR    DEFINED                 ; If x > 0 attempt to compute x**y
                07    19    0311   310          BLSS    UNDEFINED               ; Branch to error code for x < 0
             0C AC 53FD   0313   311          TSTG    exp(AP)                 ; Check sign of y (Note that x = 0)
                01    15    0317   312          BLEQ    UNDEFINED               ; Branch to error condition if y =< 0
                          0319   313
                          0319   314 ;
                          0319   315 ; If processing continues here, this implies that x = 0 and y > 0.  Return
                          0319   316 ; x**y = 0
                          0319   317 ;
                          0319   318
                   04     0319   319          RET                             ; Return
                          031A   320
                          031A   321 ;
                          031A   322 ; If processing continues here, this implies that an undefined exponentiation
                          031A   323 ; was attempted.  Signal error and return
                          031A   324 ;
                          031A   325
                          031A   326 UNDEFINED:
          50    8000 8F    3C   031A   327          MOVZWL  #^X8000, R0             ; R0/R1 <-- Reserved operand
                   51    D4   031F   328          CLRL    R1
                7E   00'8F    9A   0321   329          MOVZBL  #MTH$K_UNDEXP, -(SP)    ; Put error code on stack
       00000000'GF    01    FB   0325   330          CALLS   #1, G^MTH$$SIGNAL       ; Convert error number to 32 bit
                          032C   331                                          ;   condition code and signal error.
                          032C   332                                          ;   NOTE:  Second argument is not re-
                          032C   333                                          ;   quired since there is no JSB entry.
                   04     032C   334          RET                             ; Return
                          032D   335
                          032D   336 ;
                          032D   337 ; If processing continues here will attempt to compute x**y as 2^[y*log2(x)].
                          032D   338 ; We begin by determining k and f such that x = 2^k*f, where 1 =< f < 2.
                          032D   339 ;
                          032D   340
                          032D   341 DEFINED:
54        50    FFFF800F 8F    CB   032D   342          BICL3   #^XFFFF800F, R0, R4      ; R4 <-- 2^4*(biased exponent of x)
          54    00004010 8F    C2   0335   343          SUBL2   #^X4010, R4             ; R4 <-- 2^4*(exponent of x - 1) = 2^4*k
                     50    54    C2   033C   344          SUBL2   R4, R0                  ; R0 <-- f = 2*(fraction field of x)
                          033F   345
                          033F   346
                          033F   347 ; We are now ready to compute log2(x).  This computation is based on the
                          033F   348 ; following identity:
                          033F   349 ;
                          033F   350 ;                                          2  r---    1                        f-a
                          033F   351 ;    log2(2^k*f) = k + log2(a) + ----------- >    ------ z^(2j+1), where z = -----.
                          033F   352 ;                                 ln(2)/____ 2j+1                          f+a
                          033F   353 ;
                          033F   354 ; We begin by determining a as b^i, where b = 2^(1/32) and i is
                          033F   355 ; between 0 and 32 inclusive.  Specifically i is chosen by table look-up
```

```
                                                M 15
                     033F    356 ; in such a fashion as to minimize the magnitude of z.  Since log2(a) = i/32
                     033F    357 ; we may write
                     033F    358 ;
                     033F    359 ;                       log2(x) = k + i/32 + z*p(z*z).
                     033F    360 ;
                     033F    361
                     033F    362 EVAL_LOG2:
        5A   50   03 9C  033F    363         ROTL    #3, R0, R10
    5A  FFFFFF80 8F  CA  0343    364         BICL2   #^XFFFFFF80, R10        ; R10 <-- index to INDEX table
      5A   FCB9 CF4A 90  034A    365         MOVB    INDEX[R10], R10         ; R10 <-- i
         7E   6A44   3E  0350    366         MOVAW   (R10)[R4], -(SP)        ; (SP) <-- 2^5(k + i/32)
                     0354    367
                     0354    368 ;
                     0354    369 ; We proceed by computing z = (f-a)/(f+a).  In order to insure the accuracy of
                     0354    370 ; the final result, it is necessary to compute z to at least 68 bits. Since no
                     0354    371 ; back up data type is available, we must compute z in two parts: z = z1 + z2,
                     0354    372 ; where z1 is the high 26 bits of z and z2 is the low 53 bits of z.  Further,
                     0354    373 ; to obtain the desired accuracy it is necessary to work with a = a1 + a2,
                     0354    374 ; where a1 and a2 are the high and low 53 bits respectively of 'a'.  We begin
                     0354    375 ; computing (in single precision)
                     0354    376 ;
                     0354    377 ;                           z1 = (f - a1)/(f + a1)
                     0354    378 ;
                     0354    379 ; Note that f-a1 can be computed exactly in 53 bits, but f+a1 may require 54
                     0354    380 ; bits. The 54 bit can be determined by the exclusive or of the low bits of
                     0354    381 ; f and a1.
                     0354    382 ;
                     0354    383
      58   FD2F CF4A 7D  0354    384         MOVQ    A1_TABLE[R10], R8       ; R8/R9 <-- a1
        7E   51   59  CD  035A    385         XORL3   R9, R1, -(SP)           ; SP --> XOR of low bits of a1 and x
                     035E    386                                             ;     (This will used to determine the
                     035E    387                                             ;      54 bit of f+a1.)
        52   50   58 41FD  035E    388         ADDG3   R8, R0, R2              ; R2/R3 <-- f + a1 (rounded)
             50   58 42FD  0363    389         SUBG2   R8, R0                  ; R0/R1 <-- f - a1 (exact)
        58   50   52 47FD  0367    390         DIVG3   R2, R0, R8              ;
    59   FFFF07FF 8F   CA  036C    391         BICL2   #^XFFFF07FF, R9         ; R8/R9 <-- z1 (26 bits)
                     0373    392 ;
                     0373    393 ; To compute z2 we note
                     0373    394 ;
                     0373    395 ;      z = z1 + z2 = (f - a1 - a2)/(f + a1 + a2)
                     0373    396 ;
                     0373    397 ;         ==>   z2 = (f - a1 - a2)/(f + a1 + a2) - z1
                     0373    398 ;
                     0373    399 ; Now let v = f + a1 + a2 = v1 + v2, where v1 and v2 are the high 26 and low
                     0373    400 ; 53 bits of v respectively.  Then
                     0373    401 ;
                     0373    402 ;                   z2 = [(f - a1 - z1*v1) - (a2 + z1*v2)]/v
                     0373    403 ;
                     0373    404 ; We begin by computing v1 and f - a1 - z1*v1
                     0373    405 ;
                     0373    406
             54   52   D0  0373    407         MOVL    R2, R4                  ; R4 <-- high longword of f + a1
    55  53   FFFF07FF 8F   CB  0376    408         BICL3   #^XFFFF07FF, R3, R5     ; R4/R5 <-- high longword of f + a1 (26 bits
        56   52   54 43FD  037E    409         SUBG3   R4, R2, R6              ; R6/R7 <-- w - v1 (exact)
             54   58 44FD  0383    410         MULG2   R8, R4                  ; R4/R5 <-- z1*v1 (exact)
             50   54 42FD  0387    411         SUBG2   R4, R0                  ; R0/R1 <-- f - a1 - z1*v1 (exact)
                     038B    412 ;
```

```
                                038B    413 ; Compute v2 and a2 + a1*v2
                                038B    414 ;
          54    FE00 CF4A    7D  038B    415          MOVQ    A2_TABLE[R10], R4       ; R4/R5 <-- a2
        6E    FFFEFFFF 8F    CA  0391    416          BICL    #^XFFFEFFFF, (SP)       ; Check if w was rounded
                     06    13  0398    417          BEQL    1$                      ; Branch if not rounded
             56    FC61 CF 42FD  039A    418          SUBG2   TWO_M52, R6             ; Correct for rounding error (exact)
                  56   54 40FD  03A0    419 1$:      ADDG2   R4, R6                  ; R6/R7 <-- v2
                  56   58 44FD  03A4    420          MULG2   R8, R6                  ; R6/R7 <-- z1*v2
                  56   54 40FD  03A8    421          ADDG2   R4, R6                  ; R6/R7 <-- a2 + z1*v2
                                03AC    422 ;
                                03AC    423 ; Compute z2
                                03AC    424 ;
                  50   56 42FD  03AC    425          SUBG2   R6, R0                  ; R0/R1 <-- (f-a1-z1*v1)-(a2-z1*v2)
                  50   52 46FD  03B0    426          DIVG2   R2, R0                  ; R0/R1 <-- z2
                                03B4    427 ;
                                03B4    428 ; The next step is to compute log2(x) accurate to at least 68 bits.  This is
                                03B4    429 ; accomplished as follows, let
                                03B4    430 ;
                                03B4    431 ;                   w = 2^5*log2(x)
                                03B4    432 ;                     = (2^5)[k + i/32 + z*p(z*z)]
                                03B4    433 ;                     = 2^5*(k + i/32) + (2^5)*z*(c0 + c2*z^3 + ... + c10*z^11)
                                03B4    434 ;                     = [2^5*(k + i/32) + z'] + z'(d2*z'^2 + ... + d10*z'^10)
                                03B4    435 ;                     = [2^5*(k + i/32) + z'] + z'*q(z'*z')
                                03B4    436 ;                     = w1 + w2
                                03B4    437 ;
                                03B4    438 ; where z' = (2^5*c0)*z and w1 and w2 are the high 26 and low 53 bits of w
                                03B4    439 ; respectively.  Note that the choice of 'a' used in computing z, guarantees
                                03B4    440 ; that z' overhangs z'*q(z'*z') by at least 13 bits.  Hence, if w is computed
                                03B4    441 ; as w1 + w2, the necessary 68 bits of accuracy can be obtained.
                                03B4    442 ;
                                03B4    443 ; We begin by defining
                                03B4    444 ;
                                03B4    445 ;                   c  = high 53 bits of (2^5*c0)
                                03B4    446 ;                   c1 = high 26 bits of (2^5*c0)
                                03B4    447 ;                   c2 = low 53 bits of (2^5*c0)
                                03B4    448 ; then
                                03B4    449 ;                   z' = (z1 + z2)*(c1 + c2)
                                03B4    450 ;                      = z1*c1 + z1*c2 + z2*c.
                                03B4    451 ;
                                03B4    452
      54    58    FF07 CF 45FD  03B4    453          MULG3   C1, R8, R4              ; R4/R5 <-- c1*z1
            58    FF08 CF 44FD  03BB    454          MULG2   C2, R8                  ; R8/R9 <-- c2*z1
            50    FEF2 CF 44FD  03C1    455          MULG2   C, R0                   ; R0/R1 <-- c*z2
                  50   58 40FD  03C7    456          ADDG2   R8, R0                  ; R0/R1 <-- c*z2 + c2*z1
            58    50   54 41FD  03CB    457          ADDG3   R4, R0, R8              ; R8/R9 <-- z'
                                03D0    458
                                03D0    459 ;
                                03D0    460 ; We proceed by letting
                                03D0    461 ;
                                03D0    462 ;                   w1 = high 26 bits of 2^5*(k + i/32) + z1*c1
                                03D0    463 ; and
                                03D0    464 ;                   w2' = {[2^5*(k + i/32) + z1*c1 - w1] + z1*c2} + z2*c.
                                03D0    465 ;
                                03D0    466 ;           ==>  2^5*(k + i/32) + z' = w1 + w2'.
                                03D0    467 ;
                                03D0    468 ;           ==>  w  = [2^5*(k + i/32) + z'] + z'*q(z'*z')
                                03D0    469 ;                    = w1 + w2' + z'*q(z'*z')
```

```
                                03D0   470 ;                       = w1 + w2,
                                03D0   471 ;
                                03D0   472 ; where w2 = w2' + z'*q(z'*z')
                                03D0   473 ;
                                03D0   474
      7E    04 AE 4EFD        03D0   475          CVTLG   4(SP), -(SP)            ; (SP) <-- 2^5(k+i/32)
   56    54    6E 41FD        03D5   476          ADDG3   (SP), R4, R6            ; R6 <-- 2^5(k+i/32) + z1*c1 = w1
57    FFFF07FF 8F    CA        03DA   477          BICL2   #^XFFFF07FF, R7         ; R6/R7 <-- w1 (high 26 bits)
   52    56    8E 43FD        03E1   478          SUBG3   (SP)+, R6, R2           ; R2 <-- bits of z1*c1 included in w1
         54    52 42FD        03E6   479          SUBG2   R2, R4                  ;
   6E    50    54 41FD        03EA   480          ADDG3   R4, R0, (SP)            ; (SP) --> w2'
                                03EF   481
                                03EF   482
                                03EF   483 ;
                                03EF   484 ; Compute w2
                                03EF   485 ;
                                03EF   486
   50    58    58 45FD        03EF   487          MULG3   R8, R8, R0              ; R0/R1 <-- z'*z'
FE9D CF    03    50 55FD        03F4   488          POLYG   R0, #LOGLEN-1, LOGTAB   ; R0/R1 <-- q(z'*z')
         50    58 44FD        03FB   489          MULG2   R8, R0                  ; R0/R1 <-- z'*q(z'*z')
         50    6E 40FD        03FF   490          ADDG2   (SP), R0                ; R0/R1 <-- w2
                                0403   491
                                0403   492 ;
                                0403   493 ; We now calculate y*log2(x) = (y1+y2)*(w1+w2) = y1*w1 + y2*w1 + y*w2, where
                                0403   494 ; y1 and y2 are the high and low 26 and 27 bits of y respectively.
                                0403   495 ;
                                0403   496
                                0403   497
      54    0C AC    7D        0403   498          MOVQ    exp(AP), R4             ; R4/R5 <-- y
                                0407   499 ;
                                0407   500 ; Test for the possibility of overflow in the computation of y*w1.
                                0407   501 ; This will occur if the exponent of y plus the exponent of w1 is greater
                                0407   502 ; than 1023.
                                0407   503 ;
                                0407   504
   52    54    0B    04    EF  0407   505          EXTZV   #4, #11, R4, R2         ; biased exp of y
         52    0400 8F    A2  040C   506          SUBW2   #^X400, R2              ; unbiased exp of y
   53    56    0B    04    EF  0411   507          EXTZV   #4, #11, R6, R3         ; biased exp of w1
         53    0400 8F    A2  0416   508          SUBW2   #^X400, R3              ; unbiased exp of w1
               53    52    A0  041B   509          ADDW2   R2, R3                  ; unbiased exp of w1*y
         53    03FF 8F    B1  041E   510          CMPW    #^X3FF, R3              ; largest unbiased exp possible is 1023
                     03    18  0423   511          BGEQ    NO_SYS_OVER_FLOW
                  0094     31  0425   512          BRW     Y_TIMES_W1_OVER
                                0428   513 NO_SYS_OVER_FLOW:
               50    54 44FD  0428   514          MULG2   R4, R0                  ; R0/R1 <-- y*w2
               52    54    D0  042C   515          MOVL    R4, R2                  ; R2 <-- high longword of y1
53   55   FFFF07FF 8F    CB  042F   516          BICL3   #^XFFFF07FF, R5, R3     ; R2/R3 <-- y1
               54    52 42FD  0437   517          SUBG2   R2, R4                  ; R4/R5 <-- y2
               52    56 44FD  043B   518          MULG2   R6, R2                  ; R2/R3 <-- y1*w1
               54    56 44FD  043F   519          MULG2   R6, R4                  ; R4/R5 <-- y2*w1
                                0443   520
                                0443   521 ;
                                0443   522 ; The next step in computing 2^[y*log2(x)] is to write y*log2(x) as
                                0443   523 ;
                                0443   524 ;                   y*log2(x) = I + j/32 + g/32,
                                0443   525 ;
                                0443   526 ; where I is an integer, j is an integer between 0 and 31 inclusive, and
```

```
                                        0443   527 ; g is a fraction in the interval [-1/2, 1/2)
                                        0443   528 ;
                                        0443   529
56   52   00000000 00004358 8F 41FD     0443   530         ADDG3   #^X4358, R2, R6                ;
     56   00000000 00004358 8F 42FD     0450   531         SUBG2   #^X4358, R6                    ; R6 <-- 2^5(I + j/32)
                        52   56 42FD     045C   532         SUBG2   R6, R2                         ; R2/R3 <-- those bits of z1*y1 not
                                        0460   533                                                ;           included in 2^5(I + j/32)
                        54   52 40FD     0460   534         ADDG2   R2, R4                         ;
                        50   54 40FD     0464   535         ADDG2   R4, R0                         ; R0/R1 <-- g
          58   56   00008000 8F   CB     0468   536         BICL3   #^X00008000, R6, R8
                    58   4100 8F   B1     0470   537         CMPW    #^X4100, R8                    ; Overflow/underflow if exponent field of
                                        0475   538                                                ;   'y*2^5*log2(x)' >= 16
                             39   19     0475   539         BLSS    EXCEPTION_1                    ; Branch if 'I' is too large
                        57   56 4AFD     0477   540         CVTGL   R6, R7                         ; R7 <-- 2^5*(I + j/32) in integer
                                        047B   541
                                        047B   542 ;
                                        047B   543 ; We can now compute
                                        047B   544 ;
                                        047B   545 ;       x**y = 2^[y*log2(x)] = 2^[I + j/32 + g/32]
                                        047B   546 ;
                                        047B   547 ;            = (2^I)*[A*(B+1)] = 2^I*[A + A*B], where
                                        047B   548 ;
                                        047B   549 ; A = 2^(j/32) is obtained by table look-up and B = 2^(g/32) - 1 is obtained
                                        047B   550 ; by a Min/Max approximation.
                                        047B   551 ;
                                        047B   552
        FE4E CF   06   50 55FD          047B   553         POLYG   R0, #EXPLEN-1, EXPTAB          ; R0 <-- B =  2^(g/32) - 1. (Chebyshev
                                        0482   554                                                ;   polynomial approx. of degree 5)
52   57   FFFFFFE0 8F   CB     0482   555         BICL3   #^XFFFFFFE0, R7, R2          ; R2 <-- index into A1_TABLE
       50   FBF8 CF42 44FD     048A   556         MULG2   A1_TABLE[R2], R0             ; R0/R1 <-- A*B
       50   FCF9 CF42 40FD     0491   557         ADDG2   A2_TABLE[R2], R0             ; R0/R1 <-- A*B + A2
       50   FBEA CF42 40FD     0498   558         ADDG2   A1_TABLE[R2], R0             ; R0/R1 <-- 2^[(j+g)/32] = (A*B+A2)+A1
              57   1F   AA     049F   559         BICW2   #^X1F, R7                    ; R7 <-- 2^5*I
       57   57   FF 8F   9C     04A2   560         ROTL    #-1, R7, R7                  ; R7 <-- 2^4*I
                 50   57   A0     04A7   561         ADDW2   R7, R0                       ; R0 <-- 2^I*2^[(j+g)/32]
                 0F   50   B1     04AA   562         CMPW    R0, #^XF                     ; test for over/underflow
                      08   15     04AD   563         BLEQ    EXCEPTION_2                  ; see what exception is if neg or = 0
                           04     04AF   564 RETURN: RET                                  ; otherwise return result in R0
                                        04B0   565
                                        04B0   566 ;
                                        04B0   567 ; Handlers for software detected over/underflow conditions follow
                                        04B0   568 ;
                                        04B0   569
                                        04B0   570 EXCEPTION_1:
                        56 53FD         04B0   571         TSTG    R6                           ; if big ARG > 0 goto overflow
                        1F   18         04B3   572         BGEQ    OVER                         ;   handler, otherwise go to
                        0A   11         04B5   573         BRB     UNDER                        ;   underflow handler
                                        04B7   574
                                        04B7   575 EXCEPTION_2:
                        57 53FD         04B7   576         TSTG    R7                           ; test sign of I; if I < 0
                        18   18         04BA   577         BGEQ    OVER                         ; go to overflow handler
                                        04BC   578
                                        04BC   579 ;
                                        04BC   580 ; y*w1 would have caused a hardware system floating overflow error. If y<0,
                                        04BC   581 ; then we should return a result of 0 since result = 2^(y*(w1+w2)). Note,
                                        04BC   582 ; y can not be zero.
                                        04BC   583 ;
```

```
                          04BC    584
                          04BC    585 Y_TIMES_W1_OVER:
                    54 53FD   04BC    586         TSTG    R4                      ; if y < 0 no overflow is needed
                    13   14   04BF    587         BGTR    OVER                    ; overflow
                          04C1    588
                          04C1    589
                          04C1    590 ;
                          04C1    591 ; Underflow; if user has FU set, signal error.  Always return 0.0
                          04C1    592 ;
                          04C1    593
                          04C1    594 UNDER:
                    50   7C   04C1    595         CLRQ    R0                      ; R0/R1 <-- 0
                0B 04 AD   06   E1   04C3    596         BBC     #6, SF$W_SAVE_PSW(FP), 2$
                          04C8    597                                             ; has user enabled floating underflow?
                   7E    00'8F   9A   04C8    598         MOVZBL  #MTH$K_FLOUNDMAT, -(SP) ; Put underflow code on stack.
            00000000'GF    01   FB   04CC    599         CALLS   #1, G^MTH$$SIGNAL       ; Convert to MTH$_FLOUNDMAT (32-bit
                          04D3    600                                             ; VAX-11  exception code) and
                          04D3    601                                             ; signal condition
                    04    04D3    602 2$:     RET                                 ; return
                          04D4    603
                          04D4    604 ;
                          04D4    605 ; Signal floating overflow, return reserved operand, -0.0
                          04D4    606 ;
                          04D4    607
                          04D4    608 OVER:                                       ; else process for overflow
                   7E    00'8F   9A   04D4    609         MOVZBL  #MTH$K_FLOOVEMAT, -(SP) ; Put overflow code on stack
                50   01   0F   79   04D8    610         ASHQ    #15, #T, R0             ; R0 = result = reserved operand
                          04DC    611                                             ; -0.0. R0 will be copied to
                          04DC    612                                             ; signal mechanism vector (CHF$L_MCH_R0/R1)
                          04DC    613                                             ; so can be fixed up by any error
                          04DC    614                                             ; handler
            00000000'GF    01   FB   04DC    615         CALLS   #1, G^MTH$$SIGNAL       ; signal condition
                    04    04E3    616         RET                                 ; return - R0 restored from CHF$L_MCH_R0/R1
                          04E4    617
                          04E4    618         .END
```

```
A1_TABLE                        00000088 R      02
A2_TABLE                        00000190 R      02
ACMASK                        = 000047FC
BASE                          = 0C000004
C                               000002B8 R      02
C1                              000002C0 R      02
C2                              000002C8 R      02
COMMON                          0000030F R      02
DEFINED                         0000032D R      02
EVAL_LOG2                       0000033F R      02
EXCEPTION_1                     000004B0 R      02
EXCEPTION_2                     000004B7 R      02
EXP                           = 0000000C
EXPLEN                        = 00000007
EXPTAB                          000002D0 R      02
GET_BASE                        0000030A R      02
INDEX                           00000008 R      02
LOGLEN                        = 00000004
LOGTAB                          00000298 R      02
MTH$$SIGNAL                     ********   X    00
MTH$K_FLOOVEMAT                 ********   X    00
MTH$K_FLOUNDMAT                 ********   X    00
MTH$K_UNDEXP                    ********   X    00
NO_SYS_OVER_FLOW                00000428 R      02
OTS$POWGG                       00000308 RG     02
OVER                            000004D4 R      02
RETURN                          000004AF R      02
SF$W_SAVE_PSW                 = 00000004
TWO_M52                         00000000 R      02
UNDEFINED                       0000031A R      02
UNDER                           000004C1 R      02
Y_TIMES_W1_OVER                 000004BC 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	000004E4 (1252.)	02 (2.)	PIC USR CON REL LCL SHR EXE RD NOWRT NOVEC LONG

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

Phase	Page faults	CPU Time	Elapsed Time
Initialization	34	00:00:00.10	00:00:00.69
Command processing	109	00:00:00.53	00:00:03.41
Pass 1	126	00:00:02.13	00:00:07.64
Symbol table sort	0	00:00:00.04	00:00:00.04
Pass 2	121	00:00:01.20	00:00:04.75
Symbol table output	5	00:00:00.04	00:00:00.04
Psect synopsis output	3	00:00:00.03	00:00:00.03
Cross-reference output	0	00:00:00.00	00:00:00.00

Assembler run totals 400 00:00:04.08 00:00:16.72

The working set limit was 1200 pages.
11264 bytes (22 pages) of virtual memory were used to buffer the intermediate code.
There were 10 pages of symbol table space allocated to hold 61 non-local and 2 local symbols.
618 source lines were read in Pass 1, producing 15 object records in Pass 2.
8 pages of virtual memory were used to define 7 macros.

+----------------------------+
! 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$:UVXPOWGG/OBJ=OBJ$:UVXPOWGG MSRC$:UVXPOWGG/UPDATE=(ENH$:UVXPOWGG)

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY

OTSPOWHH
LIS

OTSPOWII
LIS

OTSPOWRJ
LIS

UUXPOWGG
LIS

OTSPOWGLU
LIS

OTSPOWHLU
LIS

OTSPOWHJ
LIS

UUXPOWCJ
LIS

UUXPOWRR
LIS

OTSPOWLUL
LIS

OTSPOWRR
LIS

OTSPOWJJ
LIS

UUXEXP
LIS

UUXGSINCO
LIS

OTSPOWRLU
LIS