MITHRIL

```
 000000   TTTTTTTTT  SSSSSSSS  PPPPPPP    000000   WW      WW HH      HH HH       HH
 000000   TTTTTTTTT  SSSSSSSS  PPPPPPP    000000   WW      WW HH      HH HH       HH
00    00     TT     SS        PP    PP  00    00   WW      WW HH      HH HH       HH
00    00     TT     SS        PP    PP  00    00   WW      WW HH      HH HH       HH
00    00     TT     SS        PP    PP  00    00   WW      WW HH      HH HH       HH
00    00     TT      SSSSSS   PPPPPPPP  00    00   WW      WW HHHHHHHHHH HHHHHHHHHH
00    00     TT      SSSSSS   PPPPPPP   00    00   WW      WW HHHHHHHHHH HHHHHHHHHH
00    00     TT          SS   PP        00    00   WW  WW  WW HH      HH HH       HH
00    00     TT          SS   PP        00    00   WW  WW  WW HH      HH HH       HH
00    00     TT          SS   PP        00    00   WWWW  WWWW HH      HH HH       HH
00    00     TT          SS   PP        00    00   WWWW  WWWW HH      HH HH       HH
 000000      TT      SSSSSSS  PP         000000   WW      WW HH      HH HH       HH
 000000      TT      SSSSSSS  PP         000000   WW      WW HH      HH HH       HH

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
```

```
0000     1              .TITLE  OTS$POWHH - REAL*16 ** REAL*16 power routine
0000     2              .IDENT  /2-006/         ; File: OTSPOWHH.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      ;
0000    30      ; FACILITY: Compiled code support library
0000    31      ;++
0000    32      ; ABSTRACT:
0000    33      ;
0000    34      ;       H_floating base to H_floating power
0000    35      ;
0000    36      ;--
0000    37      ;
0000    38      ; VERSION: 2
0000    39      ;
0000    40      ; AUTHOR:
0000    41      ;       Bob Hanek, 9-Mar-83; Version 2
0000    42      ;
```

```
0000    44              .SBTTL  HISTORY        ; Detailed current edit history
0000    45
0000    46
0000    47  ; Edit history for OTS$POWHH
0000    48  ;
0000    49  ; 2-001 Implemented new algorithm. RNH 18-Mar-83
0000    50  ; 2-002 Change references of A1_TABLE(Rx) and A2_TABLE(Rx) to A1_TABLE[Rx]
0000    51  ;       and A2_TABLE[Rx] to avoid linker problems with .ADDRESS for data.
0000    52  ;       LEB 26-May-1983
0000    53  ; 2-003 Change remaining reference of INDEX(Rx) to INDEX[Rx]. LEB 29-May-1983
0000    54  ; 2-004 Add in # signs to avoid linker errors regarding non-relocatable
0000    55  ;       references.  LEB 30-May-1983
0000    56  ; 2-005 Correct use of quadword index from INDEX.  SBL 31-May-1983
0000    57  ; 2-006 Corrected a bug involving a SYS_F_FLTOVF_F error during a
0000    58  ;       MULH2 R4, t2(SP). Code was added to see if a MTH overflow message
0000    59  ;       or a zero should be returned.  JCW 19-Jan-1984
0000    60  ;
```

OTS
Sym

A1_
A2_
ACM
BAS
C
C1
C2
DEF
EVA
EXC
EXC
EXP
EXP
EXP
IND
LOG
LOG
MTH
MTH
MTH
MTH
NO
OTS
OVE
SF$
SHI
T2
T4
T6
TAB
TWO
UND
UND
Y_T

PSE

$AB
_OT

Pha

Ini
Com
Pas
Sym
Pas
Sym

OTS$POWHH
2-006

- REAL*16 ** REAL*16 power routine   F 2   16-SEP-1984 02:00:37 VAX/VMS Macro V04-00   Page 3
DECLARATIONS   6-SEP-1984 11:28:21 [MTHRTL.SRC]OTSPOWHH.MAR;1   (3)

```
              0000    62                 .SBTTL   DECLARATIONS
              0000    63
              0000    64  ;
              0000    65  ; INCLUDE FILES:
              0000    66  ;
              0000    67
              0000    68  ;
              0000    69  ; EXTERNAL SYMBOLS:
              0000    70  ;
              0000    71
              0000    72                 .DSABL   GBL
              0000    73                 .EXTRN   MTH$$SIGNAL              ; Math error routine
              0000    74                 .EXTRN   MTH$K_FLOOVEMAT         ; Floating point overflow code
              0000    75                 .EXTRN   MTH$K_FLOUNDMAT         ; Floating point underflow code
              0000    76                 .EXTRN   MTH$K_UNDEXP            ; Undefined exponentiation code
              0000    77
              0000    78  ;
              0000    79  ; MACROS:
              0000    80  ;
              0000    81                 $SFDEF                           ; Define stack frame symbols
              0000    82  ;
              0000    83  ; EQUATED SYMBOLS:
              0000    84  ;
    00000004  0000    85                 base     = 4                    ; offset to base (by value)
    00000014  0000    86                 exp      = 20                   ; offset to exponent (by value)
              0000    87
    000003F0  0000    88                 ACMASK   = ^M<R4, R5, R6, R7, R8, R9>
              0000    89                                                 ; register saving mask
              0000    90
    00000004  0000    91                 t2       = 4                    ; Offsets from stack pointer
    00000014  0000    92                 t4       = 20                   ;    to temporary storage locations.
    00000024  0000    93                 t6       = 36                   ;    Each location is 16 bytes
              0000    94
```

```
                                    0000    96 ;
                                    0000    97 ; PSECT DECLARATIONS:
                                    0000    98 ;
                                00000000    99        .PSECT  _OTS$CODE       PIC,SHR,QUAD,EXE,NOWRT
                                    0000   100                                ; program section for OTS$ code
                                    0000   101 ;
                                    0000   102 ; CONSTANTS:
                                    0000   103 ;
                                    0000   104
                                    0000   105 ;
                                    0000   106 ; The INDEX table gives the offset (in quadwords) to the appropriate
                                    0000   107 ; entries in A1_TABLE and A2_TABLE. (NOTE: Entry 1 of the INDEX table
                                    0000   108 ; is a special encoding that is intended to access the octawords immediately
                                    0000   109 ; BEFORE the A1 and A2 tables.)
                                    0000   110 ;
                                    0000   111
        06 04 04 04 02 02 FF 00     0000   112 INDEX:  .BYTE  ^X00, ^XFF, ^X02, ^X02, ^X04, ^X04, ^X04, ^X06
        0A 0A 0A 08 08 08 06 06     0008   113         .BYTE  ^X06, ^X06, ^X08, ^X08, ^X08, ^X0A, ^X0A, ^X0A
        10 0E 0E 0E 0E 0C 0C 0C     0010   114         .BYTE  ^X0C, ^X0C, ^X0C, ^X0E, ^X0E, ^X0E, ^X0E, ^X10
        14 14 14 12 12 12 10 10     0018   115         .BYTE  ^X10, ^X10, ^X12, ^X12, ^X12, ^X14, ^X14, ^X14
        18 18 18 18 16 16 16 14     0020   116         .BYTE  ^X14, ^X16, ^X16, ^X16, ^X18, ^X18, ^X18, ^X18
        1E 1C 1C 1C 1C 1A 1A 1A     0028   117         .BYTE  ^X1A, ^X1A, ^X1A, ^X1C, ^X1C, ^X1C, ^X1C, ^X1E
        22 20 20 20 20 1E 1E 1E     0030   118         .BYTE  ^X1E, ^X1E, ^X1E, ^X20, ^X20, ^X20, ^X20, ^X22
        26 24 24 24 24 22 22 22     0038   119         .BYTE  ^X22, ^X22, ^X22, ^X24, ^X24, ^X24, ^X24, ^X26
        28 28 28 28 28 26 26 26     0040   120         .BYTE  ^X26, ^X26, ^X26, ^X28, ^X28, ^X28, ^X28, ^X28
        2C 2C 2C 2C 2A 2A 2A 2A     0048   121         .BYTE  ^X2A, ^X2A, ^X2A, ^X2A, ^X2C, ^X2C, ^X2C, ^X2C
        30 30 30 2E 2E 2E 2E 2E     0050   122         .BYTE  ^X2E, ^X2E, ^X2E, ^X2E, ^X2E, ^X30, ^X30, ^X30
        34 34 32 32 32 32 30 30     0058   123         .BYTE  ^X30, ^X30, ^X32, ^X32, ^X32, ^X32, ^X34, ^X34
        36 36 36 36 36 34 34 34     0060   124         .BYTE  ^X34, ^X34, ^X34, ^X36, ^X36, ^X36, ^X36, ^X36
        3A 3A 3A 38 38 38 38 38     0068   125         .BYTE  ^X38, ^X38, ^X38, ^X38, ^X38, ^X3A, ^X3A, ^X3A
        3C 3C 3C 3C 3C 3A 3A 3A     0070   126         .BYTE  ^X3A, ^X3A, ^X3A, ^X3C, ^X3C, ^X3C, ^X3C, ^X3C
        40 40 40 3E 3E 3E 3E 3E     0078   127         .BYTE  ^X3E, ^X3E, ^X3E, ^X3E, ^X3E, ^X40, ^X40, ^X40
                                    0080   128
                                    0080   129         .ALIGN  QUAD
                                    0080   130
                                    0080   131 ;
                                    0080   132 ; For k = 0, 1, ..., 32, the k-th entry of A1_TABLE is value of 2^(k/32)
                                    0080   133 ; rounded to 113 fraction bits and the k-th entry of A2_TABLE is the value
                                    0080   134 ; of 2(k/32) – A1_TABLE(k) rounded to 113 bits.  For k = -1, A1_TABLE gives
                                    0080   135 ; the value 2^(1/64) rounded to 113 bits and A2_TABLE give 2^(1764) –
                                    0080   136 ; A1_TABLE(-1) rounded to 113 bits
                                    0080   137 ;
                                    0080   138
7A2ACA4F F7CA0EE6 7806A3E7 02C94001 0080   139         .OCTA  ^X7A2ACA4FF7CA0EE67806A3E702C94001
                                    0090   140 A1_TABLE:
00000000 00000000 00000000 00004001 0090   141         .OCTA  ^X00000000000000000000000000004001
8CA48EB6 7C5443AE 58570D31 059B4001 00A0   142         .OCTA  ^X8CA48EB67C5443AE58570D31059B4001
42AAB718 8B92F629 989086CF 0B554001 00B0   143         .OCTA  ^X42AAB7188B92F629989086CF0B554001
318DAED9 BBF10A4E 25B51D01 11304001 00C0   144         .OCTA  ^X318DAED9BBF10A4E25B51D0111304001
B14AC50E F7C8ADCD D51783C7 172B4001 00D0   145         .OCTA  ^XB14AC50EF7C8ADCDD51783C7172B4001
0F082899 5B80A780 8B9A7316 1D484001 00E0   146         .OCTA  ^X0F0828995B80A7808B9A73161D484001
5CB6B1C1 1FAD866C 5623A6E7 23874001 00F0   147         .OCTA  ^X5CB6B1C11FAD866C5623A6E723874001
4AA4F5A2 5D1512C2 FDEEDF51 29E94001 0100   148         .OCTA  ^X4AA4F5A25D1512C2FDEEDF5129E94001
5C864630 8D5A52DE 1B71E0A3 306F4001 0110   149         .OCTA  ^X5C8646308D5A52DE1B71E0A3306F4001
47982F45 4550AA71 AA9C7373 371A4001 0120   150         .OCTA  ^X47982F454550AA71AA9C7373371A4001
D7743E13 4122235B 234264C1 3DEA4001 0130   151         .OCTA  ^XD7743E134122235B234264C13DEA4001
01A009DF 36F4D031 18928606 44E04001 0140   152         .OCTA  ^X01A009DF36F4D03118928606644E04001
```

```
2E21FEC4 397A71D4 62A2AD53 4BFD4001   0150   153        .OCTA   ^X2E21FEC4397A71D462A2AD534BFD4001
6A6449D8 A83C1DF0 D4F8B569 53424001   0160   154        .OCTA   ^X6A6449D8A83C1DF0D4F8B56953424001
EB345191 9301958C 85427DD4 5AB04001   0170   155        .OCTA   ^XEB3451919301958C85427DD045AB04001
DA436FD2 0FA04B1F A558EB03 62474001   0180   156        .OCTA   ^XDA436FD20FA04B1FA558EB0362474001
EA951366 B2FBC908 F3BCE667 6A094001   0190   157        .OCTA   ^XEA951366B2FBC908F3BCE6676A094001
ACD72EF0 370F3DD2 C5F75E8E 71F74001   01A0   158        .OCTA   ^XACD72EF0370F3DD2C5F75E8E71F74001
DA1F3F6C 51026D7D B018473E 7A114001   01B0   159        .OCTA   ^XDA1F3F6C51026D7DB018473E7A114001
4A01FAB3 F88A28AC CCE19994 82584001   01C0   160        .OCTA   ^X4A01FAB3F88A28ACCCE1999482584001
C9BBA192 7C55B5BA AACD5422 8ACE4001   01D0   161        .OCTA   ^XC9BBA1927C55B5BAAACD54228ACE4001
0A23F254 01C34F45 DC5E7B0C 93734001   01E0   162        .OCTA   ^X0A23F25401C34F45DC5E7B0C93734001
2BE6071F C46B01C7 3F09182A 9C494001   01F0   163        .OCTA   ^X2BE6071FC46B01C73F09182A9C494001
87BD1CAF 2449C8B4 E2553B23 A5504001   0200   164        .OCTA   ^X87BD1CAF2449C8B4E2553B23A5504001
205A1773 734DD5E8 AD3AF995 AE894001   0210   165        .OCTA   ^X205A1773734DD5E8AD3AF995AE894001
3C531AB5 7B086EAA B5E46F2F B7F74001   0220   166        .OCTA   ^X3C531AB57B086EAAB5E46F2FB7F74001
1BA62A09 0CB1C222 5529BDD8 C1994001   0230   167        .OCTA   ^X1BA62A090CB1C2225529BDD8C1994001
9DB71E94 3CBD9150 F9060DCE CB724001   0240   168        .OCTA   ^X9DB71E943CBD9150F9060DCECB724001
E0DDEB66 A05A725D BA488DCF D5814001   0250   169        .OCTA   ^XE0DDEB66A05A725DBA488DCFD5814001
291B39ED 8CACEB96 B9B57337 DFC94001   0260   170        .OCTA   ^X291B39ED8CACEB96B9B57337DFC94001
DB3018F5 F73A9858 490DFA2A EA4A4001   0270   171        .OCTA   ^XDB3018F5F73A9858490DFA2AEA4A4001
62BB7628 F84B0674 E45465B6 F5074001   0280   172        .OCTA   ^X62BB7628F84B0674E45465B6F5074001
00000000 00000000 00000000 00004002   0290   173        .OCTA   ^X00000000000000000000000000004002
                                      02A0   174
52A07BFC 1D910E8D 74D50A3D 085BBF8F   02A0   175        .OCTA   ^X52A07BFC1D910E8D74D50A3D085BBF8F
                                      02B0   176  A2_TABLE:
                             00000220  02B0   177                 TABLEN = <.-A1_TABLE>
                                      02B0   178        .OCTA   ^X00000000000000000000000000000000
2A10A05E AF84DF4B 95F73E34 7FE53F8D   02C0   179        .OCTA   ^X2A10A05EAF84DF4B95F73E347FE53F8D
64E815EA 7575C53D DD305BAF F26FBF8F   02D0   180        .OCTA   ^X64E815EA7575C53DDD305BAFF26FBF8F
F0008794 BD8E11A8 4E993B85 53A3BF8D   02E0   181        .OCTA   ^XF0008794BD8E11A84E993B8553A3BF8D
AF748F09 A4FC5D61 7FDFD542 E4803F8F   02F0   182        .OCTA   ^XAF748F09A4FC5D617FDFD542E4803F8F
8E44A379 3EB8A64C 46E7F77E 59D2BF8F   0300   183        .OCTA   ^X8E44A3793EB8A64C46E7F77E59D2BF8F
1A8076FE 86C84825 65BF35EA B13FBF8F   0310   184        .OCTA   ^X1A8076FE86C8482565BF35EAB13FBF8F
4E54E502 9FD7F004 DCB7BD4F 0D5DBF8F   0320   185        .OCTA   ^X4E54E5029FD7F004DCB7BD4F0D5DBF8F
46800E93 D0E560AF BC9D3D8C 2134BF8D   0330   186        .OCTA   ^X46800E93D0E560AFBC9D3D8C2134BF8D
CD90BB4E F90C1A66 AFE74945 F8F83F8F   0340   187        .OCTA   ^XCD90BB4EF90C1A66AFE74945F8F83F8F
00000344 AEEF660E F6EF1F51 174DBF8B   0350   188        .OCTA   ^X00000344AEEF660EF6EF1F51174DBF8B
33642C2C 8A8D1E9B 3D9212DE 0AC3BF8F   0360   189        .OCTA   ^X33642C2C8A8D1E9B3D9212DE0AC3BF8F
CEB04EC6 7D1BA860 EA6345D1 FC9CBF8D   0370   190        .OCTA   ^XCEB04EC67D1BA860EA6345D1FC9C3F8D
770068D8 6D73F49F 7ACFAB83 6331BF8C   0380   191        .OCTA   ^X770068D86D73F49F7ACFAB836331BF8C
8A1C8B2F EF6D81A7 4A035F20 76233F8F   0390   192        .OCTA   ^X8A1C8B2FEF6D81A74A035F2076233F8F
3B684189 D4436291 23375418 138CBF8F   03A0   193        .OCTA   ^X3B684189D443629123375418138CBF8F
642C688D 426749DD 05D4EB7B F4F83F8F   03B0   194        .OCTA   ^X642C688D426749DD05D4EB7BF4F83F8F
7698DD28 912CAEFA 930BA54E A5E5BF8E   03C0   195        .OCTA   ^X7698DD28912CAEFA930BA54EA5E5BF8E
5F186CD2 7A08814C 5DE5AD9A 7BD03F8F   03D0   196        .OCTA   ^X5F186CD27A08814C5DE5AD9A7BD03F8F
BAA0BFBF A737EA65 7581CB97 ED5A3F8C   03E0   197        .OCTA   ^XBAA0BFBFA737EA657581CB97ED5A3F8C
00707686 A36C8251 3999B0F9 F3763F8E   03F0   198        .OCTA   ^X00707686A36C82513999B0F9F3763F8E
DBC8B22F C8800BE6 F24EA853 681DBF8E   0400   199        .OCTA   ^XDBC8B22FC8800BE6F24EA853681DBF8E
338CA216 BD2E9E4D F7A386BC C894BF8F   0410   200        .OCTA   ^X338CA216BD2E9E4DF7A386BCC894BF8F
E7C82C4C 0D40F042 1FA36FEB FEBDBF8F   0420   201        .OCTA   ^XE7C82C4C0D40F0421FA36FEBFEBDBF8F
D6D0AE09 5885102C D7A9EB99 FEF03F8F   0430   202        .OCTA   ^XD6D0AE095885102CD7A9EB99FEF03F8F
691409D8 6AB32D89 D5B940F0 53223F8F   0440   203        .OCTA   ^X691409D86AB32D89D5B940F053223F8F
EA24E5CC 00109849 28D91259 9E513F8F   0450   204        .OCTA   ^XEA24E5CC0010984928D912599E513F8F
ADDC77E6 B566B74F 032D566B 87653F8F   0460   205        .OCTA   ^XADDC77E6B566B74F032D566B87653F8F
70B84139 371FE240 98FDD82A 5829BF8F   0470   206        .OCTA   ^X70B84139371FE24098FDD82A5829BF8F
38749C94 92E4ED54 BF56A513 C8963F8F   0480   207        .OCTA   ^X38749C9492E4ED54BF56A513C8963F8F
65F0BB00 3EB7EEE1 610CEA20 F86D3F8D   0490   208        .OCTA   ^X65F0BB003EB7EEE1610CEA20F86D3F8D
EC00DA6A 1DBD1759 F2B5C80E 9BFFBF87   04A0   209        .OCTA   ^XEC00DA6A1DBD1759F2B5C80E9BFFBF87
```

```
00000000 00000000 00000000 00000000   04B0   210        .OCTA   ^X00000000000000000000000000000000
                                       04C0   211
                                       04C0   212  TWO_M112:
00000000 00000000 00000000 00003F91    04C0   213        .OCTA   ^X00000000000000000000000000003F91
                                       04D0   214
D23AFDA0 7D0FE177 B82F7652 71544008    04D0   215  C:     .OCTA   ^XD23AFDA07D0FE177B82F765271544008
00000000 00000000 B82F7652 71544008    04E0   216  C1:    .OCTA   ^X0000000000000000B82F765271544008
C142AD1E FA23A474 FB41FA1F C2EE3FD7    04F0   217  C2:    .OCTA   ^XC142AD1EFA23A474FB41FA1FC2EE3FD7
                                       0500   218
                                       0500   219  LOGTAB:
E5035347 6399DAAF C2E81C2F 5E423F84    0500   220        .OCTA   ^XE50353476399DAAFC2E81C2F5E423F84   ; D16
596B3BDD 856DD665 9444039D 9D1C3F93    0510   221        .OCTA   ^X596B3BDD856DD6659444039D9D1C3F93   ; D14
4DD0503F 260881C7 BEDEBE21 F00E3FA2    0520   222        .OCTA   ^X4DD0503F260881C7BEDEBE21F00E3FA2   ; D12
738FF472 EF9D5774 8AA1F2A6 310C3FB2    0530   223        .OCTA   ^X738FF472EF9D57748AA1F2A6310C3FB2   ; D10
F2FF977E 57E375D5 D52E256F 84023FC1    0540   224        .OCTA   ^XF2FF977E57E375D5D52E256F84023FC1   ; D8
AA0C7A7A 32878429 A04E0187 03953FD1    0550   225        .OCTA   ^XAA0C7A7A32878429A04E018703953FD1   ; D6
84CFA731 F47D898B C5E241FA 7A333FE0    0560   226        .OCTA   ^X84CFA731F47D898BC5E241FA7A333FE0   ; D4
BA24049D A0F24704 C83B3FFA 47FD3FF0    0570   227        .OCTA   ^XBA24049DA0F24704C83B3FFA47FD3FF0   ; D2
00000000 00000000 00000000 00000000    0580   228        .OCTA   ^X00000000000000000000000000000000   ; D0
                            00000009    0590   229  LOGLEN = <.-LOGTAB>/16
                                       0590   230
                                       0590   231
                                       0590   232  EXPTAB:
2EE6E17A 56875E2B C298650F C3BD3F95    0590   233        .OCTA   ^X2EE6E17A56875E2BC298650FC3BD3F95   ; 12
214A5E47 57994C0F 1BB2C735 E8CA3F9F    05A0   234        .OCTA   ^X214A5E4757994C0F1BB2C735E8CA3F9F   ; 11
769A952E DA1D9F6F B8EC5158 E4CF3FA9    05B0   235        .OCTA   ^X769A952EDA1D9F6FB8EC5158E4CF3FA9   ; 10
F9095A4F A70E3DA4 5E7C3D39 B5253FB3    05C0   236        .OCTA   ^XF9095A4FA70E3DA45E7C3D39B5253FB3   ; 9
77C506DA FFE63FD8 5C82223A 62C03FBD    05D0   237        .OCTA   ^X77C506DAFFE63FD85C82223A62C03FBD   ; 8
8E69C87D 151C686B 8B0CFC58 FFCB3FC6    05E0   238        .OCTA   ^X8E69C87D151C686B8B0CFC58FFCB3FC6   ; 7
2342986B B0A876F4 6C7812F8 43093FD0    05F0   239        .OCTA   ^X2342986BB0A876F46C7812F843093FD0   ; 6
4BEC9A51 17F61107 A673FE78 5D873FD9    0600   240        .OCTA   ^X4BEC9A51117F61107A673FE785D873FD9   ; 5
EEAC0B53 CBBE729C A4E7B6FB 3B2A3FE2    0610   241        .OCTA   ^XEEAC0B53CBBE729CA4E7B6FB3B2A3FE2   ; 4
2BD32BB3 3A76F8B3 4A0B8D70 C6B03FEA    0620   242        .OCTA   ^X2BD32BB33A76F8B34A0B8D70C6B03FEA   ; 3
973606EC F16BEA86 2C58DFF8 EBFB3FF2    0630   243        .OCTA   ^X973606ECF16BEA862C58DFF8EBFB3FF2   ; 2
07E66730 93C7F357 A39E2FEF 62E43FFA    0640   244        .OCTA   ^X07E6673093C7F357A39E2FEF62E43FFA   ; 1
00000000 00000000 00000000 00000000    0650   245        .OCTA   ^X00000000000000000000000000000000
                            0000000D    0660   246  EXPLEN = <.-EXPTAB>/16
                                       0660   247
00000000 00000000 00000000 80004072    0660   248  SHIFT: .OCTA   ^X00000000000000000000000080004072
                                       0670   249
                                       0670   250        .SBTTL  OTS$POWHH_R3 - H_floating ** H_floating
                                       0670   251
                                       0670   252  ;++
                                       0670   253  ; FUNCTIONAL DESCRIPTION:
                                       0670   254  ;
                                       0670   255  ;       OTS$POWHH_R3 takes an H_floating (REAL*16) base to an
                                       0670   256  ;       H_floating power and returns an H_floating result in
                                       0670   257  ;       registers R0-R3.  This routine is for compiled code
                                       0670   258  ;       support and therefore is not required to follow the
                                       0670   259  ;       VAX Procedure Calling Standard.
                                       0670   260  ;
                                       0670   261  ;       The result of the exponentiation is:
                                       0670   262  ;
                                       0670   263  ;       base    exponent        result
                                       0670   264  ;       ----    --------        ------
                                       0670   265  ;
                                       0670   266  ;       = 0     > 0             0.0
```

```
0670   267 :      = 0      = 0                Undefined Exponentiation
0670   268 :      = 0      < 0                Undefined Exponentiation
0670   269 :
0670   270 :      < 0      any                Undefined Exponentiation
0670   271 :
0670   272 :      > 0      > 0                2^(exp * log2(base))
0670   273 :      > 0      = 0                1.0
0670   274 :      > 0      < 0                2^(exp * log2(base))
0670   275 :
0670   276 :
0670   277 :          Floating Overflow can occur.
0670   278 :          Floating Underflow can occur.
0670   279 :          Undefined Exponentiation can occur if:
0670   280 :              1)  base is 0 and exponent is 0 or negative
0670   281 :              2)  base is negative
0670   282 :
0670   283 : The basic approach to computing x**y as 2^[y*log2(x)] is the following:
0670   284 :
0670   285 :      Step 1: Compute log2(x) to sufficient precision to guarantee an
0670   286 :              accurate final result (see below.)
0670   287 :      Step 2: Compute y*log2(x) to at least the accuracy that log2(x)
0670   288 :              was computed.
0670   289 :      Step 3: Evaluate 2^[y*log2(x)] accurate to the precision of the
0670   290 :              datatype in question.
0670   291 :
0670   292 : To determine the accuracy to which log2(x) must be computed to, write
0670   293 : y*log2(x) as I + h, where I is the integer closest to y*log2(x), and
0670   294 : h = y*log2(x) - I (Note that |h| =< 1/2.)  Then
0670   295 :
0670   296 :          2^[y*log2(x)] = 2^(I + h) = (2^I)*(2^h).
0670   297 :
0670   298 : Since the factor 2^I can be incorporated into the final result by an integer
0670   299 : addition to the exponent field, we can assume that the multiplication by
0670   300 : 2^I incurs no error.  Thus the total error in the final result is determined
0670   301 : by how accurately 2^h can be computed.  If the final result has p fraction
0670   302 : bits, we would like h to have at least p good bits.  In fact it would be
0670   303 : nice if h had a few extra guard bits, say 4.  Consequently, we would like
0670   304 : h to be accurate to p + 4 bits.
0670   305 :
0670   306 : Let e be the number of bits allocated to the exponent field of the data type
0670   307 : in question.  If I requires more that e bits to represent, then overflow or
0670   308 : underflow will occur.  Therefore if the product y*log2(x) has e + p + 4 good
0670   309 : bits, the final result will be accurate.  This requires that log2(x) be
0670   310 : computed to at least p + e + 4 bits.
0670   311 :
0670   312 : Since log2(x) must be computed to more bits of precision than is available
0670   313 : in the base data type, either the next level of precision or multi-precision
0670   314 : arithmetic must be used.  We begin by writing
0670   315 :
0670   316 :
0670   317 :                            ┌---          2n+1
0670   318 :          log2(x) = log2(b) +  >   c(2n+1)*z',
0670   319 :                            /
0670   320 :                           n≡0
0670   321 :
0670   322 : Where c(1) = 1, and z' = (2/ln2)[(z-b)/(z+b)].  Hence
0670   323 :
```

OTS$POWHH
2-006

K 2

- REAL*16 ** REAL*16 power routine   16-SEP-1984 02:00:37 VAX/VMS Macro V04-00   Page 8
OTS$POWHH_R3 - H_floating ** H_floating   6-SEP-1984 11:28:21 [MTHRTL.SRC]O1SPOWHH.MAR;1   (4)

OTS
1-0

```
0670  324 :
0670  325 :                               ┌---              2n+1
0670  326 :        log2(x) = log2(b) +  z' + >  c(2n+1)*z'
0670  327 :                               /
0670  328 :                              n≡T--
0670  329 :
0670  330 :                = log2(b) + z' + p(z').
0670  331 :
0670  332 : Note that if p(z') is computed to p bits, and log2(b) + z' is computed
0670  333 : to p+e+4 bits and overhangs p(z') by e+4 bits, the required accuracy will
0670  334 : be achieved.  Consequently, the essential tricks, are to pick b such that
0670  335 : the overhang can be achieved and to compute log2(b) + z' to p + e + 4 bits.
0670  336 :
0670  337 :
0670  338 : CALLING SEQUENCE:
0670  339 :
0670  340 :     power.wh.v = OTS$POWHH_R3 (base.rh.v, exponent.rh.v)
0670  341 :
0670  342 : IMPLICIT INPUTS:
0670  343 :     none
0670  344 :
0670  345 : OUTPUT PARAMETERS:
0670  346 :     none
0670  347 :
0670  348 : IMPLICIT OUTPUTS:
0670  349 :     none
0670  350 :
0670  351 : FUNCTIONAL VALUE:
0670  352 :
0670  353 :     The H_floating result is returned in registers R0-R3.  This
0670  354 :     is a violation of the VAX procedure calling standard but is
0670  355 :     allowed for compiled code support routines.
0670  356 :
0670  357 : SIDE EFFECTS:
0670  358 :
0670  359 :     Modifies registers R0-R3!
0670  360 :     MTH$K_FLOOVEMAT - Floating overflow
0670  361 :     MTH$K_FLOUNDMAT - Floating underflow if FU bit is set
0670  362 :     SS$_ROPRAND - Reserved operand fault
0670  363 :     SIGNALs MTH$_UNDEXP (82 = ' UNDEFINED EXPONENTIATION') if
0670  364 :             1)  Base is 0 and exponent is 0 or negative
0670  365 :             2)  base is negative
0670  366 :
0670  367 :
0670  368 :--
```

```
                         03F0   0670   370           .ENTRY  OTS$POWHH_R3, ^M<>
                                0672   371
                                0672   372 ;
                                0672   373 ; Move x to R0/R3.  If x < 0, or x = 0 and y =< 0, return "UNDEFINED
                                0672   374 ; EXPONENTIATION" error condition, otherwise attempt to compute x**y
                                0672   375 ;
                                0672   376
              5E   34     C2    0672   377           SUBL    #52, SP                 ; Allocate space on the stack
          50   04 AC   70FD    0675   378           MOVH    base(AP), R0            ; R0/R3 <-- x
                    1D     14    067A   379           BGTR    DEFINED                ; If x > 0 attempt to compute x**y
                    07     19    067C   380           BLSS    UNDEFINED              ; Branch to error code for x < 0
              14 AC   73FD    067E   381           TSTH    exp(AP)                ; Check sign of y (Note that x = 0)
                    01     15    0682   382           BLEQ    UNDEFINED              ; Branch to error condition if y =< 0
                                0684   383
                                0684   384 ;
                                0684   385 ; If processing continues here, this implies that x = 0 and y > 0.  Return
                                0684   386 ; x**y = 0
                                0684   387 ;
                                0684   388
                         04    0684   389           RET                             ; Return
                                0685   390
                                0685   391 ;
                                0685   392 ; If processing continues here, this implies that an undefined exponentiation
                                0685   393 ; was attempted.  Signal error and return
                                0685   394 ;
                                0685   395
                                0685   396 UNDEFINED:
              50   7CFD    0685   397           CLRO    R0                      ;
          50   8000 8F     B0    0688   398           MOVW    #^X8000, R0            ; R0/R3 <-- Reserved operand
          7E     00'8F     9A    068D   399           MOVZBL  #MTH$K_UNDEXP, -(SP)   ; Put error code on stack
    00000000'GF     01     FB    0691   400           CALLS   #1, G^MTH$$SIGNAL       ; Convert error number to to 32 bit
                                0698   401                                           ;   condition code and signal error.
                                0698   402                                           ;   NOTE:  Second argument is not re-
                                0698   403                                           ;   quired since there is no JSB entry.
                         04    0698   404           RET                             ; Return
                                0699   405
                                0699   406 ;
                                0699   407 ; If processing continues here will attempt to compute x**y as 2^[y*log2(x)].
                                0699   408 ; We begin by determining k and f such that x = 2^k*f, where 1 =< f < 2.
                                0699   409 ;
                                0699   410
                                0699   411 DEFINED:
      54   50   FFFF8000 8F    CB    0699   412           BICL3   #^XFFFF8000, R0, R4    ; R4 <-- biased exponent of x
      54   00004001 8F    C2    06A1   413           SUBL    #^X4001, R4            ; R4 <-- k = exponent of x - 1
              50   54     C2    06A8   414           SUBL    R4, R0                 ; R0 <-- f = 2*(fraction field of x)
                                06AB   415
                                06AB   416 ;
                                06AB   417 ; We are now ready to compute log2(x).  This computation is based on the
                                06AB   418 ; following identity:
                                06AB   419 ;
                                06AB   420 ;                                 2  ,---      1                        f-a
                                06AB   421 ;   log2(2^k*f) = k + log2(a) + ----- >     ---- z^(2j+1), where z = -----.
                                06AB   422 ;                               ln(2)/____   2j+1                       f+a
                                06AB   423 ;
                                06AB   424 ; We begin by determining a as b^i, where b = 2^(1/64) and i is 0, 2, 4, ...
                                06AB   425 ; 64  or 1. Specifically i is chosen by table look-up in such a fashion as
                                06AB   426 ; to minimize the magnitude of z.  Since log2(a) = i/64  we may write
```

M 2

OTS$POWHH   - REAL*16 ** REAL*16 power routine   16-SEP-1984 02:00:37   VAX/VMS Macro V04-00   Page 10
2-006   OTS$POWHH_R3 - H_floating ** H_floating   6-SEP-1984 11:28:21   [MTHRTL.SRC]OTSPOWHH.MAR;1   (5)

OTS
1-0

```
                          06AB   427 ;
                          06AB   428 ;                          log2(x) = k + i/64 + z*p(z*z).
                          06AB   429 ;
                          06AB   430 ; NOTE: For i = 2, 4, ..., 64, we may write i = 2n, and hence i/64 = n/32, i.e.
                          06AB   431 ; a is an integral power or 2^(1/32). These values are stored in A1_TABLE and
                          06AB   432 ; A2_TABLE. For i = 1, the value of 2^(1/64) is stored immediately BEFORE
                          06AB   433 ; A1_TABLE and A2_TABLE. Consequently, to access the value of 2^(1/64) from
                          06AB   434 ; the table, a negative index is used.
                          06AB   435 ;
                          06AB   436
                          06AB   437 EVAL_LOG2:
      55    50    07    9C  06AB   438         ROTL    #7, R0, R5              ; R5(0:6) <-- high 7 fraction bits of f
      54    54    06    78  06AF   439         ASHL    #6, R4, R4              ; R4 <-- 2^6*k
 55   FFFFFF80 8F  CA  06B3   440         BICL    #^XFFFFFF80, R5         ; R5 <-- index to INDEX table
 55   FFFFF93F EF45 98  06BA   441         CVTBL   L^INDEX[R5], R5         ; R5 <-- i or -1
            06    18  06C2   442         BGEQ    1$                      ; Branch if i
            54    D6  06C4   443         INCL    R4                      ; R4 <-- 2^6(k + 1/64)
            55    D7  06C6   444         DECL    R5                      ; R5 <-- -2
            03    11  06C8   445         BRB     2$                      ; Join common code
      54    55    C0  06CA   446 1$:     ADDL    R5, R4                  ; R5 <-- 2^6*(k + i/64)
                          06CD   447
                          06CD   448 ;
                          06CD   449 ; Since there is no back up data type to compute the necessary guard bits, we
                          06CD   450 ; proceed by computing z = (f-a)/(f+a) in two parts: z = z1 + z2, where z1 is
                          06CD   451 ; the high 53 bits of z and z2 is the low 113 bits of z. Further, to obtain
                          06CD   452 ; the desired accuracy it is necessary to work with a = a1 + a2, where a1 and
                          06CD   453 ; a2 are the high and low 113 bits respectively of a. We begin
                          06CD   454 ; computing (in G-format)
                          06CD   455 ;
                          06CD   456 ;                          z1 = (f - a1)/(f + a1)
                          06CD   457 ;
                          06CD   458 ; Note that f-a1 can be computed exactly in 113 bits, but f+a1 may require 114
                          06CD   459 ; bits. The 114th bit can be determined by the exclusive or of the low bits of
                          06CD   460 ; f and a1.
                          06CD   461 ;
                          06CD   462
 55   55  FF 8F  78  06CD   463 2$:     ASHL    #-1, R5, R5             ; R5 <--- octaword offset into A1, A2_TABLE
 56   FFFFF9B6 EF45 7DFD 06D2 464       MOVO    L^A1_TABLE[R5], R6      ; R6/R9 <-- a1
      6E    53    59  CD  06DB 465       XORL3   R9, R3, (SP)           ; SP --> XOR of low bits of a1 and x
                          06DF   466                                       (This will be used to determine
                          06DF   467                                        the 114th bit of f+a1.)
 04 AE  50  56 63FD  06DF   468         SUBH3   R6, R0, t2(SP)         ; t2 <-- f - a1 (exact)
 14 AE  50  56 61FD  06E5   469         ADDH3   R6, R0, t4(SP)         ; t4 <-- f + a1 (rounded)
       56  14 AE 76FD 06EB  470         CVTHG   t4(SP), R6             ; R6/R7 <-- f + a1
       58  04 AE 76FD 06F0  471         CVTHG   t2(SP), R8             ; R8/R9 <-- f - a1
       58  56  46FD  06F5   472         DIVG2   R6, R8                 ; R8/R9 <-- z1 (G)
    24 AE 58  56FD   06F9   473         CVTGH   R8, t6(SP)             ; t6 <-- z1 (H)
                          06FE   474
                          06FE   475 ;
                          06FE   476 ; To compute z2 we note
                          06FE   477 ;
                          06FE   478 ;     z = z1 + z2 = (f - a1 - a2)/(f + a1 + a2)
                          06FE   479 ;
                          06FE   480 ;     ==>   z2 = (f - a1 - a2)/(f + a1 + a2) - z1
                          06FE   481 ;
                          06FE   482 ; Now let v = f + a1 + a2 = v1 + v2, where v1 and v2 are the high 49 and low
                          06FE   483 ; 113 bits of v respectively. Then
```

OTS$POWHH
2-006

N 2
- REAL*16 ** REAL*16 power routine   16-SEP-1984 02:00:37 VAX/VMS Macro V04-00   Page 11
OTS$POWHH_R3 - H_floating ** H_floating   6-SEP-1984 11:28:21 [MTHRTL.SRC]OTSPOWHH.MAR;1   (5)

OTS
1-0

```
                              06FE   484 ;
                              06FE   485 ;                               z2 = [((f - a1 - z1*v1) - (a2 + z1*v2)]/v
                              06FE   486 ;
                              06FE   487 ; We begin by computing v1 and f - a1 - z1*v1
                              06FE   488 ;
                              06FE   489 ;
           56    14 AE    7D  06FE   490         MOVQ    t4(SP), R6              ; R6/R7 <-- high quadword of f + a1
                  58    00 7D  0702   491         MOVQ    #0, R8                 ; R6/R9 <-- v1
        50   14 AE   56 63FD  0705   492         SUBH3   R6, t4(SP), R0         ; R0/R3 <-- w - v1 (exact)
           56    24 AE 64FD  070B   493         MULH2   t6(SP), R6             ; R6/R9 <-- z1*v1 (exact)
        04 AE    56 62FD      0710   494         SUBH2   R6, t2(SP)             ; t2 <-- f - a1 - z1*v1 (exact)
                              0715   495 ;
                              0715   496 ; Compute v2 and a2 + a1*v2
                              0715   497 ;
     6E   FFFEFFFF 8F    CA  0715   498         BICL    #^XFFFEFFFF, (SP)      ; Check if w was rounded
              06    13        071C   499         BEQL    3$                     ; Branch if not rounded
        50    FD9D CF 62FD    071E   500         SUBH2   TWO_M112, R0           ; Correct for rounding error (exact)
     50  FFFFFB84 EF45 60FD  0724   501 3$:     ADDH2   L^A2_TABLE[R5], R0     ; R0/R3 <-- v2
        50    24 AE 64FD      072D   502         MULH2   t6(SP), R0             ; R0/R3 <-- z1*v2
     50  FFFFFB76 EF45 60FD  0732   503         ADDH2   L^A2_TABLE[R5], R0     ; R0/R3 <-- a2 + z1*v2
                              073B   504 ;
                              073B   505 ; Compute z2
                              073B   506 ;
        04 AE    50 62FD      073B   507         SUBH2   R0, t2(SP)             ; t2 <-- (f-a1-z1*v1)-(a2-z1*v2)
     56  04 AE   14 AE 67FD  0740   508         DIVH3   t4(SP), t2(SP), R6     ; R6/R9 <-- z2
                              0747   509 ;
                              0747   510 ; The next step is to compute log2(x) accurate to at least 128 bits.  This is
                              0747   511 ; accomplished as follows, let
                              0747   512 ;
                              0747   513 ;              w = 2^6*log2(x)
                              0747   514 ;                = (2^6)[k + i/64 + z*p(z*z)]
                              0747   515 ;                = 2^6(k + i/64) + (2^6)*z*(c0 + c2*z^3 + ... + c10*z^11)
                              0747   516 ;                = [2^6*(k + i/64) + z'] + z'(d2*z'^2 + ... + d10*z'^10)
                              0747   517 ;                = [2^6*(k + i/64) + z'] + z'*q(z'*z')
                              0747   518 ;                = w1 + w2
                              0747   519 ;
                              0747   520 ; where z' = (2^6*c0)*z and w1 and w2 are the high 49 and low 113 bits of w
                              0747   521 ; respectively.  Note that the choice of 'a' used in computing z, guarantees
                              0747   522 ; that z' overhangs z'*q(z'*z') by at least 15 bits.  Hence, if w is computed
                              0747   523 ; as w1 + w2, 128 bits of accuracy can be obtained.
                              0747   524 ;
                              0747   525 ; We begin by defining
                              0747   526 ;
                              0747   527 ;              c  = high 113 bits of (2^6*c0)
                              0747   528 ;              c1 = high 49 bits of (2^6*c0)
                              0747   529 ;              c2 = low 113 bits of (2^6*c0)
                              0747   530 ; then
                              0747   531 ;              z' = (z1 + z2)*(c1 + c2)
                              0747   532 ;                 = z1*c1 + z1*c2 + z2*c.
                              0747   533 ;
                              0747   534 ;
     50   24 AE  FDA4 CF 65FD 0747   535         MULH3   C2, t6(SP), R0         ; R0/R3 <-- c2*z1
          24 AE  FDBC CF 64FD 074F   536         MULH2   C1, t6(SP)             ; t6 <-- c1*z1
          56   FD75 CF 64FD  0756   537         MULH2   C, R6                  ; R6/R7 <-- c*z2
        14 AE   56   50 61FD  075C   538         ADDH3   R0, R6, t4(SP)         ; t4 <-- c*z2 + c2*z1
     56  14 AE   24 AE 61FD  0762   539         ADDH3   t6(SP), t4(SP), R6     ; R6/R9 <-- z'
                              0769   540
```

OTS$POWHH
2-0`6

B 3
- REAL*16 ** REAL*16 power routine   16-SEP-1984 02:00:37 VAX/VMS Macro V04-00   Page 12
OTS$POWHH_R3 - H_floating ** H_floating   6-SEP-1984 11:28:21 [MTHRTL.SRC]OTSPOWHH.MAR;1   (5)

OTS$
1-0(

```
                        0769   541  ;
                        0769   542  ; We proceed by letting
                        0769   543  ;
                        0769   544  ;              w1 = high 49 bits of 2^6*(k + i/32) + z1*c1
                        0769   545  ; and
                        0769   546  ;              w2' = ([2^6*(k + i/32) + z1*c1 - w1] + z1*c2) + z2*c.
                        0769   547  ;
                        0769   548  ;      ==>  2^6*(k + i/64) + z' = w1 + w2'.
                        0769   549  ;
                        0769   550  ;      ==>  w  = [2^6*(k + i/64) + z'] + z'*q(z'*z')
                        0769   551  ;                = w1 + w2' + z'*q(z'*z')
                        0769   552  ;                = w1 + w2,
                        0769   553  ;
                        0769   554  ; where w2 = w2' + z'*Q(z'*z')
                        0769   555  ;
                        0769   556  ;
            54    54 4EFD 0769   557         CVTLG    R4, R4              ; R4/R5 <-- 2^6(k + i/64)
         52    24 AE 76FD 076D   558         CVTHG    t6(SP), R2          ; R2/R3 <-- z1*c1
      50   52    54 41FD 0772   559         ADDG3    R4, R2, R0          ; R0/R1 <-- 2^6(k+i/32) + z1*c1
      52   50    54 43FD 0777   560         SUBG3    R4, R0, R2          ; R2/R3 <-- bits of z1*c1 included in w1
         52    52 56FD 077C   561         CVTGH    R2, R2              ; R2/R5 <-- bits of z1*c1 included in w1
      24 AE    52 62FD 0780   562         SUBH2    R2, t6(SP)          ; [2^6(k+i/32)-w1+z1*c2]
   14 AE   24 AE 60FD 0785   563         ADDH2    t6(SP), t4(SP)      ; t4 <-- w2'
      04 AE    50 56FD 078B   564         CVTGH    R0, t2(SP)          ; t2 <-- w1
                        0790   565
                        0790   566  ;
                        0790   567  ; Compute w2
                        0790   568  ;
                        0790   569
      50    56    56 65FD 0790   570         MULH3    R6, R6, R0          ; R0/R3 <-- z'*z'
   FD64 CF   08    50 75FD 0795   571         POLYH    R0, #LOGLEN-1, LOGTAB  ; R0/R3 <-- q(z'*z')
            50    56 64FD 079C   572         MULH2    R6, R0              ; R0/R3 <-- z'*Q(z'*z')
         14 AE    50 60FD 07A0   573         ADDH2    R0, t4(SP)          ; t4 <-- w2
                        07A5   574
                        07A5   575  ;
                        07A5   576  ; We now calculate y*log2(x) = (y1+y2)*(w1+w2) = y1*w1 + y2*w1 + y*w2, where
                        07A5   577  ; y1 and y2 are the high 56 and low 57 bits of y respectively.
                        07A5   578  ;
                        07A5   579
      50    14 AC 70FD 07A5   580         MOVH     exp(AP), R0         ; R0/R3 <-- y
                        07AA   581  ;
                        07AA   582  ; Test for the possibility of overflow in the computation of y*w1.
                        07AA   583  ; This will occur if the exponent of y plus the exponent of w1 is greater
                        07AA   584  ; than 16383.
                        07AA   585  ;
                        07AA   586
   54    50 FFFF8000 8F  CB 07AA   587         BICL3    #^XFFFF8000, R0, R4    ; biased exp of y
      54    4000 8F  A2 07B2   588         SUBW2    #^X4000, R4            ; unbiased exp of y
55   04 AE FFFF8000 8F  CB 07B7   589         BICL3    #^XFFFF8000, t2(SP), R5  ; biased exp of y
      55    4000 8F  A2 07C0   590         SUBW2    #^X4000, R5            ; unbiased exp of y
         55    54  A0 07C5   591         ADDW2    R4, R5                ; unbiased exp of w1*y
      55    3FFF 8F  B1 07C8   592         CMPW     #^X3FFF, R5           ; largest unbiased exp possible is 16383
            03  18 07CD   593         BGEQ     NO_SYS_OVER_FLOW
            0093  31 07CF   594         BRW      Y_TIMES_W1_OVER
                        07D2   595 NO_SYS_OVER_FLOW:
      14 AE    50 64FD 07D2   596         MULH2    R0, t4(SP)            ; t4 <-- y*w2
         54    50  7D 07D7   597         MOVQ     R0, R4                ; R4/R5 <-- high 49 bits of y
```

```
 56   52   FFFF01FF 8F   CB  07DA  598         BICL3   #^XFFFF01FF, R2, R6    ; R4/R6 <-- high 56 bits of y
            57   00   D0  07E2  599         MOVL    #0, R7                 ; R4/R7 <-- y1
            50   54 62FD  07E5  600         SUBH2   R4, R0                 ; R0/R3 <-- y2
        50   04 AE 64FD  07E9  601         MULH2   t2(SP), R0             ; R0/R3 <-- y2*w1
        04 AE   54 64FD  07EE  602         MULH2   R4, t2(SP)             ; t2 <-- y1*w1
                          07F3  603
                          07F3  604  :
                          07F3  605  : The next step in computing 2^[y*log2(x)] is to write y*log2(x) as
                          07F3  606  :
                          07F3  607  :              y*log2(x) = I + j/32 + g/32,
                          07F3  608  :
                          07F3  609  : where I is an integer, j is an integer between 0 and 31 inclusive, and
                          07F3  610  : g is a fraction in the interval [-1/2, 1/2)
                          07F3  611  :
                          07F3  612
 54   14 AE   04 AE 61FD  07F3  613         ADDH3   t2(SP), t4(SP), R4     ; R4/R7 <-- y1*w1 + y*w2
        54   FE61 CF 60FD  07FA  614         ADDH2   SHIFT, R4
        54   FE5B CF 62FD  0800  615         SUBH2   SHIFT, R4              ; R4/R7 <-- 2^6(I + j/32)
        04 AE   54 62FD  0806  616         SUBH2   R4, t2(SP)             ; t2 <-- those bits of z1*y1 not
                          080B  617                                       ;             included in 2^6(I + j/32)
        50   04 AE 60FD  080B  618         ADDH2   t2(SP), R0
        50   14 AE 60FD  0810  619         ADDH2   t4(SP), R0             ; R0/R3 <-- 2^7(g/32)
 58   54   8000 8F   AB  0815  620         BICW3   #^X8000, R4, R8        ; R8 <-- exponent field of 2^6(I+j/32)
            54   D7  081B  621         DECL    R4                     ; R4 <-- 2^5*y*log2(x)
 58   4013 8F   B1  081D  622         CMPW    #^X4013, R8
            36   15  0822  623         BLEQ    EXCEPTION_1
        58   54 6AFD  0824  624         CVTHL   R4, R8                 ; R8 <-- 2^5*(I + j/32) in integer
                          0828  625
                          0828  626  :
                          0828  627  : We can now compute
                          0828  628  :
                          0828  629  :     x**y = 2^[y*log2(x)] = 2^[I + j/32 + g/32]
                          0828  630  :
                          0828  631  :          = (2^I)*[A*(B+1)] = 2^I*[A + A*B], where
                          0828  632  :
                          0828  633  : A = 2^(j/32) is obtained by table look-up and B = 2^(g/32) - 1 is obtained
                          0828  634  : by a Min/Max approximation.
                          0828  635  :
                          0828  636
 FD61 CF   0C   50 75FD  0828  637         POLYH   R0, #EXPLEN-1, EXPTAB  ; R0/R3 <-- B = 2^(g/32) - 1
 59   58   FFFFFFE0 8F   CB  082F  638         BICL3   #^XFFFFFFE0, R8, R9    ; R9 <-- index into A1_TABLE
        59   F853 CF 7EFD  0837  639         MOVAO   A1_TABLE[R9], R9       ; R9 <-- address of A
            50   69 64FD  083E  640         MULH2   (R9), R0               ; R0/R3 <-- A*B
        50   0220 C9 60FD  0842  641         ADDH2   TABLEN(R9), R0         ; R0/R3 <-- A*B + A2
            50   69 60FD  0848  642         ADDH2   (R9), R0               ; R0/R3 <-- 2^[(j+g)/32] = (A*B+A2)+A1
        58   1F   AA  084C  643         BICW    #^X1F, R8              ; R7 = 2^5*I
 58   58   FB 8F   9C  084F  644         ROTL    #-5, R8, R8
        50   58   A0  0854  645         ADDW    R8, R0                 ; R0/R3 <-- 2^I*2^[(j+g)/32]
            08   15  0857  646         BLEQ    EXCEPTION_2            ; see what exception is if neg or = 0
            04  0859  647         RET                                ; otherwise return result in R0
                          085A  648
                          085A  649  :
                          085A  650  : Handlers for software detected over/underflow conditions follow
                          085A  651  :
                          085A  652
                          085A  653  EXCEPTION_1:
        50 73FD  085A  654         TSTH    R0                     ; if big ARG > 0 goto overflow
```

OTS$POWHH
2-006

            D 3
- REAL*16 ** REAL*16 power routine   16-SEP-1984 02:00:37 VAX/VMS Macro V04-00   Page 14
OTS$POWHH_R3 - H_floating ** H_floating   6-SEP-1984 11:28:21 [MTHRTL.SRC]OTSPOWHH.MAR;1   (5)

```
          1F   18  085D   655          BGEQ     OVER                    ;    handler, otherwise go to
          09   11  085F   656          BRB      UNDER                   ;    underflow handler
                   0861   657
                   0861   658 EXCEPTION_2:
          57   B5  0861   659          TSTW     R7                      ; test sign of I; if I < 0
          19   18  0863   660          BGEQ     OVER                    ; go to overflow handler
                   0865   661 ;
                   0865   662 ; y*w1 would have caused a hardware system floating overflow error. If y<0,
                   0865   663 ; then we should return a result of 0 since result = 2^(y*(w1+w2)). Note,
                   0865   664 ; y can not be zero.
                   0865   665 ;
                   0865   666
                   0865   667 Y_TIMES_W1_OVER:
          50 73FD  0865   668          TSTH     R0                      ; if y < 0 no overflow is needed
          14   14  0868   669          BGTR     OVER                    ; overflow
                   086A   670
                   086A   671 ;
                   086A   672 ; Underflow; if user has FU set, signal error.  Always return 0.0
                   086A   673 ;
                   086A   674
                   086A   675 UNDER:
          50 7CFD  086A   676          CLRO     R0                      ; R0/R3 <-- 0
    0B 04 AD  06  E1 086D   677          BBC      #6, SF$W_SAVE_PSW(FP), 2$
                   0872   678                                           ; has user enabled floating underflow?
       7E  00'8F  9A 0872   679          MOVZBL   #MTH$K_FLOUNDMAT, -(SP) ; Put underflow code on stack
                   0876   680                                           ; convert to MTH$_FLOUNDMAT (32-bit VAX-11
                   0876   681                                           ; exception code)
  00000000'GF  01  FB 0876   682          CALLS    #1, G^MTH$$SIGNAL       ; signal condition
                   04  087D   683 2$:      RET                          ; return
                   087E   684
                   087E   685 ;
                   087E   686 ; Signal floating overflow, return reserved operand, -0.0
                   087E   687 ;
                   087E   688
       7E  00'8F  9A 087E   689 OVER:    MOVZBL   #MTH$K_FLOOVEMAT, -(SP) ; Put overflow code on stack
          50 7CFD  0882   690          CLRO     R0                      ; R0 = result = reserved operand
    50   8000 8F  B0 0885   691          MOVW     #^X8000, R0             ; -0.0. R0 will be copied to
                   088A   692                                           ; signal mechanism vector (CHF$L_MCH_R0/R1)
                   088A   693                                           ; so can be fixed up by any error
                   088A   694                                           ; handler
  00000000'GF  01  FB 088A   695          CALLS    #1, G^MTH$$SIGNAL       ; signal condition,
                   04  0891   696          RET                          ; return - R0 restored from CHF$L_MCH_R0/R1
                   0892   697
                   0892   698          .END
```

E 3

| OTS$POWHH | - REAL*16 ** REAL*16 power routine | 16-SEP-1984 02:00:37 VAX/VMS Macro V04-00 | Page 15 | **F |
| Symbol table | | 6-SEP-1984 11:28:21 [MTHRTL.SRC]OTSPOWHH.MAR;1 | (5) | |

```
A1_TABLE            00000090 R      02
A2_TABLE            000002B0 R      02
ACMASK           = 000003F0
BASE             = 00000004
C                  000004D0 R      02
C1                 C00004E0 R      02
C2                 000004F0 R      02
DEFINED            00000699 R      02
EVAL_LOG2          000006AB R      02
EXCEPTION_1        0000085A R      02
EXCEPTION_2        00000861 R      02
EXP              = 00000014
EXPLEN           = 0000000D
EXPTAB             00000590 R      02
INDEX              00000000 R      02
LOGLEN           = 00000009
LOGTAB             00000500 R      02
MTH$$SIGNAL        ******** X      00
MTH$K_FLOOVEMAT    ******** X      00
MTH$K_FLOUNDMAT    ******** X      00
MTH$K_UNDEXP       ******** X      00
NO_SYS_OVER_FLOW   000007D2 R      02
OTS$POWHH_R3       0C000670 RG     02
OVER               0000087E R      02
SF$W_SAVE_PSW    = 00000004
SHIFT              00000660 R      02
T2               = 00000004
T4               = 00000014
T6               = 00000024
TABLEN           = 00000220
TWO_M112           000004C0 R      02
UNDEFINED          00000685 R      02
UNDER              0000086A R      02
Y_TIMES_W1_OVER    00000865 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	00000892 (	2194.)	02 (	2.)	PIC	USR	CON	REL	LCL	SHR	EXE	RD	NOWRT	NOVEC	QUAD

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

Phase	Page faults	CPU Time	Elapsed Time
Initialization	33	00:00:00.10	00:00:00.84
Command processing	115	00:00:00.58	00:00:02.96
Pass 1	136	00:00:02.71	00:00:07.82
Symbol table sort	0	00:00:00.05	00:00:00.07
Pass 2	136	00:00:01.65	00:00:05.64
Symbol table output	6	00:00:00.03	00:00:00.22

F 3

OTS$POWHH   - REAL*16 ** REAL*16 power routine   16-SEP-1984 02:00:37 VAX/VMS Macro V04-00   Page 16   OIS
VAX-11 Macro Run Statistics   6-SEP-1984 11:28:21 [MTHRTL.SRC]OTSPOWHH.MAR;1   (5)   Tab

```
Psect synopsis output              2      00:00:00.04     00:00:00.04
Cross-reference output             0      00:00:00.00     00:00:00.00
Assembler run totals             430      00:00:05.16     00:00:17.64
```

The working set limit was 1200 pages.
13981 bytes (28 pages) of virtual memory were used to buffer the intermediate code.
There were 10 pages of symbol table space allocated to hold 62 non-local and 4 local symbols.
758 source lines were read in Pass 1, producing 17 object records in Pass 2.
9 pages of virtual memory were used to define 8 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$:OTSPOWHH/OBJ=OBJ$:OTSPOWHH MSRC$:MTHJACKET/UPDATE=(ENH$:MTHJACKET)+MSRC