


```

000000  TTTTTTTTTT  SSSSSSSS  PPPPPPPP  000000  WW  WW  DDDDDDDD  DDDDDDDD
000000  TTTTTTTTTT  SSSSSSSS  PPPPPPPP  000000  WW  WW  DDDDDDDD  DDDDDDDD
00 00  TT  SS  SS  SS  PP  PP  PP  00 00  WW  WW  DD  DD  DD  DD  DD
00 00  TT  SS  SS  SS  PP  PP  PP  00 00  WW  WW  DD  DD  DD  DD  DD
00 00  TT  SS  SS  SS  PP  PP  PP  00 00  WW  WW  DD  DD  DD  DD  DD
00 00  TT  SS  SS  SS  PP  PP  PP  00 00  WW  WW  DD  DD  DD  DD  DD
00 00  TT  SS  SS  SS  PP  PP  PP  00 00  WW  WW  DD  DD  DD  DD  DD
00 00  TT  SS  SS  SS  PP  PP  PP  00 00  WW  WW  DD  DD  DD  DD  DD
00 00  TT  SS  SS  SS  PP  PP  PP  00 00  WW  WW  DD  DD  DD  DD  DD
00 00  TT  SS  SS  SS  PP  PP  PP  00 00  WW  WW  DD  DD  DD  DD  DD
000000  TT  SSSSSSSS  PP  000000  WW  WW  DDDDDDDD  DDDDDDDD
000000  TT  SSSSSSSS  PP  000000  WW  WW  DDDDDDDD  DDDDDDDD

```

```

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

```

(2) 58
(2) 88
(4) 220

HISTORY ; Detailed current edit history
DECLARATIONS
OTSS\$POWDD - DOUBLE to DOUBLE giving DOUBLE result

```

0000 1      .TITLE  OTSSPOWDD - DOUBLE PRECISION ** DOUBLE PRECISION power routine
0000 2      .IDENT  /2-007/ ; File: OTSSPOWDD.MAR Edit: JCW2007
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 AN
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 : 1) Double base to double power.
0000 34 : 2) Floating base to double power.
0000 35 : 3) Double base to floating power.
0000 36
0000 37 : Floating overflow can occur
0000 38 : Undefined exponentiation can occur if:
0000 39 : 1) Negative base
0000 40 : 2) 0 base and power is 0 or negative.
0000 41
0000 42
0000 43
0000 44 :--
0000 45
0000 46 : VERSION: 2
0000 47
0000 48 : HISTORY:
0000 49
0000 50 : AUTHOR:
0000 51 : Bob Hanek, 3-Mar-83: Version 2
0000 52
0000 53 : MODIFIED BY:
0000 54
0000 55
0000 56

```

```
0000 58      .SBTTL HISTORY      ; Detailed current edit history
0000 59
0000 60
0000 61 : Edit history for Version 2 of OTSSPOWDD
0000 62 :
0000 63 : 2-001 Implemented new algorithm. RNH 3-Mar-83
0000 64 : 2-002 The code would overflow when underflow was the proper result. This
0000 65 : error arose from the ADDF3 #^X4DC0, R2, R6 code, which rounded the
0000 66 : result so that when SUBF #^X4DC0, R6 was performed the value in
0000 67 : R6 was greater than R2, causing R0 to be positive when it should have
0000 68 : been negative. The bugfix was to change the test for overflow from
0000 69 : R0 to R6 (both of these should be of the same sign). JCW 13-MAY-83.
0000 70 : 2-003 Change INDEX table to be local instead of global. Also change
0000 71 : the instruction format from INDEX(Rx) to INDEX[Rx] to avoid linker
0000 72 : errors. LEB 25-May-1983
0000 73 : 2-004 Change remaining table references to use [Rx] instead of (Rx).
0000 74 : Make A1 TABLE and A2_TABLE local symbols instead of globals.
0000 75 : LEB 26-May-1983
0000 76 : 2-005 Change leftover table reference to use [Rx] instead of (Rx).
0000 77 : LEB 29-May-1983
0000 78 : 2-006 Added two ROTL #-3,Rx,Rx instructions to scale the value of Rx back
0000 79 : from 'index*2^3' to 'index' before A_TABLE[Rx] is referenced. The
0000 80 : INDEX was not scaled back to yield values of 'index' instead of
0000 81 : 'index*2^3' because the mathematics of the code used does need the
0000 82 : value of index*2^3 in several computations. JCW 7-Jun-1983
0000 83 : 2-007 Corrected a bug involving a SYS_F_FLTOVF_F error during a MULR R6, R2.
0000 84 : Code was added to see if a MTH overflow message or a zero should be
0000 85 : returned. JCW 19-Jan-1984
0000 86 :
0000 87 :
```

```

0000 88      .SBTTL  DECLARATIONS
0000 89
0000 90      :
0000 91      : INCLUDE FILES:
0000 92      :
0000 93      :
0000 94      :
0000 95      :
0000 96      : EXTERNAL SYMBOLS:
0000 97      :
0000 98      .DSABL  GBL
0000 99      .EXTRN  MTH$$SIGNAL      ; Math error routine
0000 100     .EXTRN  MTH$K_FLOOVEMAT  ; Floating overflow code
0000 101     .EXTRN  MTH$K_FLOUNDMAT ; Floating underflow code
0000 102     .EXTRN  MTH$K_UNDEXP    ; Undefined exponentiation code
0000 103
0000 104     :
0000 105     : MACROS:
0000 106     :
0000 107     .SSFDEF                    ; Define stack frame symbols
0000 108     :
0000 109     : EQUATED SYMBOLS:
0000 110     :
00000004 0000 111     base   = 4          ; base input formal - by-value
0000000C 0000 112     exp    = 12         ; exponent input formal - by-value
00000008 0000 113     fexp   = 8          ; exponent when base is floating
000007FC 0000 114     ACMASK = ^M< R2, R3, R4, R5, R6, R7, R8, R9, R10> ; register saving mask
0000 115

```

OTS
Syn
A1
A2
ACF
BAS
C
C1
C2
COM
DEF
EVA
EXC
EXC
EXP
EXP
EXP
FEX
GET
INC
LOC
LOC
MTH
MTH
MTH
OTS
OTS
OTS
OVE
RET
SFS
TWO
UND
UND
Y_T

PSE

\$AE
_01

Ph

In
Con
Pas
Syn
Pas
Syn

```

0000 117 :
0000 118 : PSECT DECLARATIONS:
0000 119 :
0000 120 : .PSECT _OTSS$CODE          PIC,SHR,QUAD,EXE,NOVRT
0000 121 :                               ; program section for OTSS$ code
0000 122 :
0000 123 : CONSTANTS:
0000 124 :
0000 125 :
0000 126 :
0000 127 : The INDEX table gives the appropriate byte offset into the A1 and A2 Tables.
0000 128 :
0000 129 :
08 08 08 08 08 00 00 00 0000 130 INDEX: .BYTE ^X00, ^X00, ^X00, ^X08, ^X08, ^X08, ^X08, ^X08
18 10 10 10 10 10 10 08 0008 131 .BYTE ^X08, ^X10, ^X10, ^X10, ^X10, ^X10, ^X10, ^X18
20 20 20 18 18 18 18 18 0010 132 .BYTE ^X18, ^X18, ^X18, ^X18, ^X18, ^X20, ^X20, ^X20
28 28 28 28 20 20 20 20 0018 133 .BYTE ^X20, ^X20, ^X20, ^X20, ^X28, ^X28, ^X28, ^X28
30 30 30 30 30 30 28 28 0020 134 .BYTE ^X28, ^X28, ^X30, ^X30, ^X30, ^X30, ^X30, ^X30
38 38 38 38 38 38 30 30 0028 135 .BYTE ^X30, ^X30, ^X38, ^X38, ^X38, ^X38, ^X38, ^X38
40 40 40 40 40 40 40 38 0030 136 .BYTE ^X38, ^X40, ^X40, ^X40, ^X40, ^X40, ^X40, ^X40
48 48 48 48 48 48 48 40 0038 137 .BYTE ^X40, ^X48, ^X48, ^X48, ^X48, ^X48, ^X48, ^X48
50 50 50 50 50 50 50 48 0040 138 .BYTE ^X48, ^X50, ^X50, ^X50, ^X50, ^X50, ^X50, ^X50
58 58 58 58 58 58 50 50 0048 139 .BYTE ^X50, ^X50, ^X58, ^X58, ^X58, ^X58, ^X58, ^X58
60 60 60 60 60 58 58 58 0050 140 .BYTE ^X58, ^X58, ^X58, ^X60, ^X60, ^X60, ^X60, ^X60
68 68 68 68 60 60 60 60 0058 141 .BYTE ^X60, ^X60, ^X60, ^X60, ^X68, ^X68, ^X68, ^X68
70 70 68 68 68 68 68 68 0060 142 .BYTE ^X68, ^X68, ^X68, ^X68, ^X68, ^X68, ^X70, ^X70
70 70 70 70 70 70 70 70 0068 143 .BYTE ^X70, ^X70, ^X70, ^X70, ^X70, ^X70, ^X70, ^X70
78 78 78 78 78 78 78 78 0070 144 .BYTE ^X78, ^X78, ^X78, ^X78, ^X78, ^X78, ^X78, ^X78
80 80 80 80 80 78 78 78 0078 145 .BYTE ^X78, ^X78, ^X78, ^X80, ^X80, ^X80, ^X80, ^X80
0080 146
0080 147
0080 148 .ALIGN QUAD
0080 149
0080 150
0080 151 : For k = 0, 1, ..., 16, the k-th entry of the A1 table is 2^(k/16) rounded
0080 152 : to 56 bits and the k-th entry of the A2 table 2^(k/16) - A1_TABLE(k)
0080 153 : rounded to 56 bits.
0080 154 :
0080 155
0080 156 A1_TABLE:
00000000 00004080 0080 157 .QUAD ^X00000000000004080
487B67CC AAC34085 0088 158 .QUAD ^X487B67CCAAC34085
8BD7E3EA 95C1408B 0090 159 .QUAD ^X8BD7E3EA95C1408B
11C373AB C3D34091 0098 160 .QUAD ^X11C373ABC3D34091
B8A9518D 37F04098 00A0 161 .QUAD ^XB8A9518D37F04098
A1126091 F532409E 00A8 162 .QUAD ^XA1126091F532409E
5139A9B1 FED640A5 00B0 163 .QUAD ^X5139A9B1FED640A5
A14BEA42 583E40AD 00B8 164 .QUAD ^XA14BEA42583E40AD
DE6533F9 04F340B5 00C0 165 .QUAD ^XDE6533F904F340B5
0C379F58 08A340BD 00C8 166 .QUAD ^X0C379F5808A340BD
06DB1155 672A40C5 00D0 167 .QUAD ^X06DB1155672A40C5
8481151F 248C40CE 00D8 168 .QUAD ^X8481151F248C40CE
9D6BCAD6 44FC40D7 00E0 169 .QUAD ^X9D6BCAD644FC40D7
94E1EC2A CCDE40E0 00E8 170 .QUAD ^X94E1EC2ACCDE40E0
2439E7DD C0C640EA 00F0 171 .QUAD ^X2439E7DDC0C640EA
86CC1524 257D40F5 00F8 172 .QUAD ^X86CC1524257D40F5
00000000 00004100 0100 173 .QUAD ^X00000000000004100

```

```

0108 174
0108 175 A2_TABLE:
00000000 00000000 0108 176 .QUAD ^X0000000000000000
0AA6DC61 2E4A2326 0110 177 .QUAD ^X0AA6DC612E4A2326
3AD6EBC5 20DCA348 0118 178 .QUAD ^X3AD6EBC520DCA348
B96B6382 3F5B23D8 0120 179 .QUAD ^XB96B63823F5B23D8
5C864630 8D5A245E 0128 180 .QUAD ^X5C8646308D5A245E
288CC1EC BEDDA424 0130 181 .QUAD ^X288CC1ECBEDDA424
477C04EF 1A14A32F 0138 182 .QUAD ^X477C04EF1A14A32F
29975CDC D9FDA3E6 0140 183 .QUAD ^X29975CDCD9FDA3E6
156BEC99 4D04A477 0148 184 .QUAD ^X156BEC994D04A477
25E1C093 AEFDA402 0150 185 .QUAD ^X25E1C093AEFDA402
6C8ABCDA 0754A38B 0158 186 .QUAD ^X6C8ABCDA0754A38B
506AE383 EE53A360 0160 187 .QUAD ^X506AE383EE53A360
FD2C4466 6597A2BC 0168 188 .QUAD ^XFD2C44666597A2BC
6E9AA824 32C42308 0170 189 .QUAD ^X6E9AA82432C42308
C1B9D6CD 40B523BB 0178 190 .QUAD ^XC1B9D6CD40B523BB
B66031EB EE7423B1 0180 191 .QUAD ^XB66031EBEE7423B1
00000000 00000000 0188 192 .QUAD ^X0000000000000000
0190 193
00000000 00002500 0190 194 TWO_M55: .QUAD ^X00000000000002500
0198 195
17F1295C AA3B4488 0198 196 C: .QUAD ^X17F1295CAA3B4488
00002000 AA3B4488 01A0 197 C1: .QUAD ^X00002000AA3B4488
E8800BBB C17F3695 01A8 198 C2: .QUAD ^XE8800BBBC17F3695
01B0 199
973BB7EA 01121CC2 01B0 200 LOGTAB: .QUAD ^X973BB7EA01121CC2
2742C3D0 CA802581 01B8 201 .QUAD ^X2742C3D0CA802581
F145FD62 19A02E3D 01C0 202 .QUAD ^XF145FD6219A02E3D
1DA4FD64 FE9F3723 01C8 203 .QUAD ^X1DA4FD64FE9F3723
00000000 00000000 01D0 204 .QUAD ^X0000000000000000
00000005 01D8 205 LOGLEN = <.-LOGTAB>/8
01D8 206
01D8 207
01D8 208 EXPTAB:
0D2F11A3 E6F41FFF 01D8 209 .QUAD ^X0D2F11A3E6F41FFF : 0.27094695278739704E-19
2D1FB089 85242521 01E0 210 .QUAD ^X2D1FB08985242521 : 0.35024086731698588E-16
6DC43BFA C3FF2A2E 01E8 211 .QUAD ^X6DC43BFAC3FF2A2E : 0.38805761563274748E-13
D9857D84 955B2F1D 01F0 212 .QUAD ^XD9857D84955B2F1D : 0.35830323049894278E-10
0627B825 584633E3 01F8 213 .QUAD ^X0627B825584633E3 : 0.26466421444330986E-07
2C9CFC16 FDEF3875 0200 214 .QUAD ^X2C9CFC16FDEF3875 : 0.14662262387640433E-04
CF7AF7D1 72173CB1 0208 215 .QUAD ^XCF7AF7D172173CB1 : 0.54152123481245728E-02
00000000 00000000 0210 216 .QUAD ^X0000000000000000 : 0.0
00000008 0218 217 EXPLEN = <.-EXPTAB>/8
0218 218

```


0218 220 .SBTTL OTSS\$POWDD - DOUBLE to DOUBLE giving DOUBLE result

0218 221
 0218 222 :++
 0218 223 : FUNCTIONAL DESCRIPTION:

0218 224 :
 0218 225 : OTSS\$POWDR - DOUBLE result = DOUBLE base ** FLOATING exponent
 0218 226 : OTSS\$POWRD - DOUBLE result = FLOATING base ** DOUBLE exponent
 0218 227 : OTSS\$POWDD - DOUBLE result = DOUBLE base ** DOUBLE exponent

0218 228 :
 0218 229 : The DOUBLE result is given by:

base	exponent	result
----	-----	-----
= 0	> 0	0.0
= 0	= 0	Undefined Exponentiation
= 0	< 0	Undefined Exponentiation
< 0	any	Undefined Exponentiation
> 0	> 0	2^(exp * LOG2(base))
> 0	= 0	1.0
> 0	< 0	2^(exp * LOG2(base))

0218 243 :
 0218 244 : Floating Overflow and Underflow can occur.
 0218 245 : Undefined Exponentiation can occur if:
 0218 246 : 1) base is 0 and exponent is 0 or negative
 0218 247 : 2) base is negative

0218 248 :
 0218 249 : The basic approach to computing x**y as 2^[y*log2(x)] is the following:

- 0218 250 :
 0218 251 : Step 1: Compute log2(x) to sufficient precision to guarantee an
 0218 252 : accurate final result (see below.)
 0218 253 : Step 2: Compute y*log2(x) to at least the accuracy that log2(x)
 0218 254 : was computed.
 0218 255 : Step 3: Evaluate 2^[y*log2(x)] accurate to the precision of the
 0218 256 : datatype in question.
 0218 257 :

0218 258 :
 0218 259 : To determine the accuracy to which log2(x) must be computed to, write
 0218 260 : y*log2(x) as I + h, where I is the integer closest to y*log2(x), and
 0218 261 : h = y*log2(x) - I (Note that |h| =< 1/2.) Then

0218 262 :
 0218 263 :
$$2^{[y \cdot \log_2(x)]} = 2^{(I + h)} = (2^I) \cdot (2^h).$$

0218 264 :
 0218 265 : Since the factor 2^I can be incorporated into the final result by an integer
 0218 266 : addition to the exponent field, we can assume that the multiplication by
 0218 267 : 2^I incurs no error. Thus the total error in the final result is determined
 0218 268 : by how accurately 2^h can be computed. If the final result has p fraction
 0218 269 : bits, we would like h to have at least p good bits. In fact it would be
 0218 270 : nice if h had a few extra guard bits, say 4. Consequently, we would like
 0218 271 : h to be accurate to p + 4 bits.

0218 272 :
 0218 273 : Let e be the number of bits allocated to the exponent field of the data type
 0218 274 : in question. If I requires more than e bits to represent, then overflow or
 0218 275 : underflow will occur. Therefore if the product y*log2(x) has e + p + 4 good
 0218 276 : bits, the final result will be accurate. This requires that log2(x) be

0218 277 : computed to at least $p + e + 4$ bits.
 0218 278 :
 0218 279 : Since $\log_2(x)$ must be computed to more bits of precision than is available
 0218 280 : in the base data type, either the next level of precision or multi-precision
 0218 281 : arithmetic must be used. We begin by writing

$$\log_2(x) = \log_2(b) + \sum_{n=0}^{\infty} c(2n+1)z'^{2n+1}$$

0218 282 :
 0218 283 :
 0218 284 :
 0218 285 :
 0218 286 :
 0218 287 :
 0218 288 :
 0218 289 : Where $c(1) = 1$, and $z' = (2/\ln 2)[(z-b)/(z+b)]$. Hence

$$\begin{aligned} \log_2(x) &= \log_2(b) + z' + \sum_{n=1}^{\infty} c(2n+1)z'^{2n+1} \\ &= \log_2(b) + z' + p(z'). \end{aligned}$$

0218 290 :
 0218 291 :
 0218 292 :
 0218 293 :
 0218 294 :
 0218 295 :
 0218 296 :
 0218 297 :
 0218 298 :
 0218 299 : Note that if $p(z')$ is computed to p bits, and $\log_2(b) + z'$ is computed
 0218 300 : to $p+e+4$ bits and overhangs $p(z')$ by $e+4$ bits, the required accuracy will
 0218 301 : be achieved. Consequently, the essential tricks, are to pick b such that
 0218 302 : the overhang can be achieved and to compute $\log_2(b) + z'$ to $p + e + 4$ bits.

0218 303 :
 0218 304 :
 0218 305 : CALLING SEQUENCE:

0218 306 :
 0218 307 : power.wd.v = OTSSPOWDR (base.rd.v, exponent.rf.v)
 0218 308 : power.wd.v = OTSSPOWRD (base.rf.v, exponent.rd.v)
 0218 309 : power.wd.v = OTSSPOWDD (base.rd.v, exponent.rd.v)

0218 310 :
 0218 311 : INPUT PARAMETERS:

0218 312 : Base and exponent parameters are call by value

0218 313 :
 0218 314 : IMPLICIT INPUTS:

0218 315 : none

0218 316 :
 0218 317 : OUTPUT PARAMETERS:

0218 318 : none

0218 319 :
 0218 320 : IMPLICIT OUTPUTS:

0218 321 : none

0218 322 :
 0218 323 : FUNCTIONAL VALUE:

0218 324 : OTSSPOWDR - DOUBLE base ** FLOATING power
 0218 325 : OTSSPOWRD - FLOATING base ** DOUBLE power
 0218 326 : OTSSPOWDD - DOUBLE base ** DOUBLE power

0218 327 :
 0218 328 : SIDE EFFECTS:

0218 329 :
 0218 330 : SIGNALS MTH\$K_FLOOVEMAT if floating overflow.
 0218 331 : SIGNALS MTH\$K_FLOUNDMAT if floating underflow.
 0218 332 : SIGNALS MTH\$ ONDEXP (82 = 'UNDEFINED EXPONENTIATION') if
 0218 333 : 1) Base is 0 and exponent is 0 or negative

G-S\$POWDD
2-007

- DOUBLE PRECISION ** DOUBLE PRECISION p 16-SEP-1984 01:57:20 VAX/VMS Macro V04-00 Page 8
OTSS\$POWDD - DOUBLE to DOUBLE giving DOUB 6-SEP-1984 11:28:03 [MTHRTL.SRC]OTSPowDD.MAR;1 (4)

0218 334 :
0218 335 :
0218 336 :
0218 337 :--

2) base is negative

OT:
1-

```

07FC 0218 339
      021A 340
      SE 14 C2 021A 341
      6E 02 D0 021D 342
04 AE 04 AC 70 0220 343
OC AE 0C AC 56 0225 344
0000024C'EF 6E FA 022A 345
      04 0231 346
      0232 347
      0232 348
07FC 0232 349
      0234 350
      SE 14 C2 0234 351
      6E 02 D0 0237 352
04 AE 04 AC 56 023A 353
OC AE 08 AC 70 023F 354
0000024C'EF 6E FA 0244 355
      04 024B 356
      024C 357

```

```

.ENTRY OTSS$POWDR, ACMASK ;
SUBL #20, SP ; Allocate 5 longwords on the stack
MOVL #2, (SP) ; Set number of arguments equal to 2
MOVD base(AP), base(SP) ; Move base to stack
CVTFD exp(AP), exp(SP) ; convert FLOATING exponent to DOUBLE
CALLG (SP), OTSS$POWDD ; Call OTSS$POWDD
RET

```

```

.ENTRY OTSS$POWRD, ACMASK ;
SUBL #20, SP ; Allocate 5 longwords on the stack
MOVL #2, (SP) ; Set number of arguments equal to 2
CVTFD base(AP), base(SP) ; convert FLOATING base to DOUBLE
MOVD fexp(AP), exp(SP) ; Move exponent to stack
CALLG (SP), OTSS$POWDD ; Call OTSS$POWDD
RET

```

```

07FC 024C 359
      024C 360      .ENTRY OTS$POWDD, ACMASK      ;
      024E 361      :
      024E 362      : Move x to R0/R1.  If x < 0, or x = 0 and y =< 0, return 'UNDEFINED
      024E 363      : EXPONENTIATION' error condition, otherwise attempt to compute x**y
      024E 364      :
      024E 365      :
      024E 366      GET_BASE:
50  04  AC  70  024E 367      MOVD      base(AP), R0      ; R0/R1 <-- x
      0252 368      COMMON:
      0252 369      BGTR      DEFINED      ; If x > 0 attempt to compute x**y
      0254 370      BLSS      UNDEFINED     ; Branch to error code for x < 0
      0C  AC  73  0256 371      TSTD      exp(AP)      ; Check sign of y (Note that x = 0)
      01  15  0259 372      BLEQ      UNDEFINED     ; Branch to error condition if y =< 0
      025B 373      :
      025B 374      :
      025B 375      : If processing continues here, this implies that x = 0 and y > 0.  Return
      025B 376      : x**y = 0
      025B 377      :
      04  025B 378      :
      025B 379      RET      ; Return
      025C 380      :
      025C 381      :
      025C 382      : If processing continues here, this implies that an undefined exponentiation
      025C 383      : was attempted.  Signal error and return
      025C 384      :
      025C 385      :
      025C 386      UNDEFINED:
      50  8000 8F  3C  025C 387      MOVZWL  #^X8000, R0      ; R0/R1 <-- Reserved operand
      0261 388      CLRL      R1
      7E  00'8F  9A  0263 389      MOVZBL  #MTH$K UNDEXP, -(SP) ; Put error code on stack
00000000'GF  01  FB  0267 390      CALLS   #1, G^MTH$S$SIGNAL ; Convert error number to 32 bit
      026E 391      : condition code and signal error.
      026E 392      : NOTE: Second argument is not re-
      026E 393      : quired since there is no JSB entry.
      04  026E 394      RET      ; Return
      026F 395      :
      026F 396      :
      026F 397      : If processing continues here will attempt to compute x**y as 2^[y*log2(x)].
      026F 398      : We begin by determining k and f such that x = 2^k*f, where 1 =< f < 2.
      026F 399      :
      026F 400      :
      54  50  FFFF807F 8F  CB  026F 401      DEFINED:
      54  54  00004080 8F  C2  0277 403      SUBL      #^X4080, R4      ; R4 <-- 2^7*(biased exponent of x)
      50  54  C2  027E 404      SUBL      R4, R0      ; R4 <-- 2^7*k = 2^7*(exponent of x - 1)
      0281 405      : R0 <-- f = 2*(fraction field of x)
      0281 406      :
      0281 407      : We are now ready to compute log2(x).  This computation is based on the
      0281 408      : following identity:
      0281 409      :
      0281 410      :
      0281 411      : 
$$\log_2(2^k * f) = k + \log_2(a) + \frac{2^{-j}}{\ln(2)} > \frac{1}{2j+1} z^{(2j+1)}, \text{ where } z = \frac{f-a}{f+a}.$$

      0281 412      :
      0281 413      :
      0281 414      : We begin by determining a as b^i, where b = 2^(1/16) and i is
      0281 415      : between 0 and 16 inclusive.  Specifically i is chosen by table look-up
    
```

```

0281 416 : in such a fashion as to minimize the magnitude of z. Since log2(a) = i/16
0281 417 : we may write
0281 418 :
0281 419 :           log2(x) = k + i/16 + z*p(z*z).
0281 420 :
0281 421 :
0281 422 EVAL_LOG2:
SA 50  FFFFFFFF80 8F  CB 0281 423  BICL3  #^FFFFFFF80, R0, R10  : R10 <-- index to INDEX table
      SA  FD72 CF4A  90 0289 424  MOVB   INDEX[R10], R10    : R10 <-- i*2^3
      7E  54  SA  C1 028F 425  ADDL3  R10, R4, -(SP)     : SP --> 2^7*(k + i/16)
      SA  SA  FD 8F  9C 0293 426  ROTL   #-3, R10, R10     : R10 <-- i
0298 427 : R10 will be multiplied by 2^3 by
0298 428 : table references like the line below.
0298 429 : The linker will cause an error if
0298 430 : ( ) are used instead of [ ] for these
0298 431 : table references.
0298 432 :
0298 433 :
0298 434 : We proceed by computing z = (f-a)/(f+a). In order to insure the accuracy of
0298 435 : the final result, it is necessary to compute z to at least 68 bits. Since no
0298 436 : back up data type is available, we must compute z in two parts: z = z1 + z2,
0298 437 : where z1 is the high 24 bits of z and z2 is the low 56 bits of z. Further,
0298 438 : to obtain the desired accuracy it is necessary to work with a = a1 + a2,
0298 439 : where a1 and a2 are the high and low 56 bits respectively of 'a'. We begin
0298 440 : computing (in single precision)
0298 441 :
0298 442 :           z1 = (f - a1)/(f + a1)
0298 443 :
0298 444 : Note that f-a1 can be computed exactly in 56 bits, but f+a1 may require 57
0298 445 : bits. The 57 bit can be determined by the exclusive or of the low bits of
0298 446 : f and a1.
0298 447 :
0298 448 :
0298 449           MOVQ   A1_TABLE[R10], R8      : R8/R9 <-- a1
0298 450           XORL3  R9, R1, -(SP)         : SP --> XOR of low bits of a1 and x
0298 451 :           :           (This will be used to determine the
0298 452 :           :           57 bit of f+a1.)
0298 453           ADDD3  R8, R0, R2            : R2/R3 <-- f + a1 (rounded)
0298 454           SUBD   R8, R0              : R0/R1 <-- f - a1 (exact)
0298 455           DIVF3  R2, R0, R8         : R8 <-- z1 (single)
0298 456           MOVL   #0, R9            : R8/R9 <-- z1 (double)
0280 457 :
0280 458 : To compute z2 we note
0280 459 :
0280 460 :           z = z1 + z2 = (f - a1 - a2)/(f + a1 + a2)
0280 461 :
0280 462 :           ==> z2 = (f - a1 - a2)/(f + a1 + a2) - z1
0280 463 :
0280 464 : Now let v = f + a1 + a2 = v1 + v2, where v1 and v2 are the high 24 and low
0280 465 : 56 bits of v respectively. Then
0280 466 :
0280 467 :           z2 = [(f - a1 - z1*v1) - (a2 + z1*v2)]/v
0280 468 :
0280 469 : We begin by computing v1 and f - a1 - z1*v1
0280 470 :
0280 471 :
0280 472           MOVL   R2, R4                : R4 <-- high longword of f + a1
  
```

54 52 D0

```

56 55 00 D0 02B3 473      MOVL  #0, R5          ; R4/R5 <-- v1
    52 54 63 02B6 474      SUBD3  R4, R2, R6     ; R6/R7 <-- f + a1 - v1 (exact)
    54 58 64 02BA 475      MULD  R8, R4          ; R4/R5 <-- z1*v1 (exact)
    50 54 62 02BD 476      SUBD  R4, R0          ; R0/R1 <-- f - a1 - z1*v1 (exact)
    02C0 477 :
    02C0 478 : Compute v2 and a2 + a1*v2
    02C0 479 :
54 FE43 CF4A 7D 02C0 480      MOVQ  A2, TABLE[R10], R4 ; R4/R5 <-- a2
6E FFFEFF 8F CA 02C6 481      BICL  #^XFFFFFFFF, (SP) ; Check if w was rounded
    05 13 02CD 482      BEQL  1$ ; Branch if not rounded
56 FEBD CF 62 02CF 483      SUBD  TWO_M55, R6 ; Correct for rounding error (exact)
    56 54 60 02D4 484 1$: ADDD  R4, R6 ; R6/R7 <-- v2
    56 58 64 02D7 485      MULD  R8, R6 ; R6/R7 <-- z1*v2
    56 54 60 02DA 486      ADDD  R4, R6 ; R6/R7 <-- a2 + z1*v2
    02DD 487 :
    02DD 488 : Compute z2
    02DD 489 :
    50 56 62 02DD 490      SUBD  R6, R0 ; R0/R1 <-- (f-a1-z1*v1)-(a2-z1*v2)
    50 52 66 02E0 491      DIVD  R2, R0 ; R0/R1 <-- z2
    02E3 492 :
    02E3 493 : The next step is to compute log2(x) accurate to at least 68 bits. This is
    02E3 494 : accomplished as follows, let
    02E3 495 :
    02E3 496 :  $w = 2^{*7} * \log_2(x)$ 
    02E3 497 :  $= (2^{*7}) [k + i/16 + z * p(z * z)]$ 
    02E3 498 :  $= 2^{*7} * (k + i/16) + (2^{*7}) * z * (c_0 + c_2 * z^2 + \dots + c_{10} * z^{*10})$ 
    02E3 499 :  $= [2^{*7} * (k + i/16) + z'] + z' * (d_2 * z'^2 + \dots + d_{10} * z'^{*10})$ 
    02E3 500 :  $= [2^{*7} * (k + i/16) + z'] + z' * q(z' * z')$ 
    02E3 501 :  $= w_1 + w_2$ 
    02E3 502 :
    02E3 503 : where  $z' = (2^{*7} * c_0) * z$  and  $w_1$  and  $w_2$  are the high 24 and low 56 bits of  $w$ 
    02E3 504 : respectively. Note that the choice of 'a' used in computing  $z$ , guarantees
    02E3 505 : that  $z'$  overhangs  $z' * q(z' * z')$  by at least 13 bits. Hence, if  $w$  is computed
    02E3 506 : as  $w_1 + w_2$ , the necessary 68 bits of accuracy can be obtained.  $c_0 = 2^{*6} / \ln(2)$ .
    02E3 507 :
    02E3 508 : We begin by defining
    02E3 509 :
    02E3 510 :  $c =$  high 56 bits of  $(2^{*7} * c_0)$ 
    02E3 511 :  $c_1 =$  high 28 bits of  $(2^{*7} * c_0)$ 
    02E3 512 :  $c_2 =$  low 56 bits of  $(2^{*7} * c_0)$ 
    02E3 513 : then
    02E3 514 :  $z' = (z_1 + z_2) * (c_1 + c_2)$ 
    02E3 515 :  $= z_1 * c_1 + z_1 * c_2 + z_2 * c.$ 
    02E3 516 :
54 58 FEB9 CF 65 02E3 518      MULD3 C1, R8, R4 ; R4/R5 <-- c1*z1
    58 FEBB CF 64 02E9 519      MULD  C2, R8 ; R8/R9 <-- c2*z1
    50 FEA6 CF 64 02EE 520      MULD  C, R0 ; R0/R1 <-- c*z2
    50 58 60 02F3 521      ADDD  R8, R0 ; R0/R1 <-- c*z2 + c2*z1
    58 50 54 61 02F6 522      ADDD3 R4, R0, R8 ; R8/R9 <-- z'
    02FA 523 :
    02FA 524 : We proceed by letting
    02FA 525 :
    02FA 526 :  $w_1 =$  high 24 bits of  $2^{*7} * (k + i/16) + z_1 * c_1$ 
    02FA 527 : and
    02FA 528 :  $w_2' = \{(2^{*7} * (k + i/16) + z_1 * c_1 - w_1) + z_1 * c_2\} + z_2 * c.$ 
    02FA 529 :

```

```

02FA 530 :
02FA 531 : ==> 2^7*(k + i/16) + z' = w1 + w2'.
02FA 532 :
02FA 533 : ==> w = [2^7*(k + i/16) + z'] + z'*q(z'*z')
02FA 534 :         = w1 + w2' + z'*q(z'*z')
02FA 535 :         = w1 + w2,
02FA 536 :
02FA 537 : where w2 = w2' + z'*q(z'*z')
02FA 538 :
02FA 539 :
5A 04 AE 4E 02FA 540 CVTLF 4(SP), R10 ; R10 <-- 2^7(k + i/16)
56 54 5A 41 02FE 541 ADDF3 R10, R4, R6 ; R6 <-- 2^7(k+i/16) + z1*c1 in single
57 00 D0 0302 542 MOVL #0, R7 ; R6/R7 <-- w1
52 56 5A 43 0305 543 SUBF3 R10, R6, R2 ; R2 <-- bits of z1*c1 included in w1
; = -[2(k+i/16)+z1*c1 - w1]
53 00 D0 0309 544 MOVL #0, R3 ; Convert R2 to double
54 52 62 030C 545 SUBD R2, R4 ; [2^7(k+i/16)+z1*c1 - w1] + c1*z1
6E 50 54 61 030F 546 ADDD3 R4, R0, (SP) ; (SP) --> w2'
0313 548 :
0313 549 :
0313 550 : Compute w2
0313 551 :
0313 552 :
FE93 50 58 58 65 0313 553 MULD3 R8, R8, R0 ; R0/R1 <-- z'*z'
CF 04 50 75 0317 554 POLYD R0, #LOGLEN-1, LOGTAB ; R0/R1 <-- q(z'*z')
50 58 64 031D 555 MULD R8, R0 ; R0/R1 <-- z'*q(z'*z')
50 6E 60 0320 556 ADDD (SP), R0 ; R0/R1 <-- w2
0323 557 :
0323 558 :
0323 559 : We now calculate y*log2(x) = (y1+y2)*(w1+w2) = y1*w1 + y2*w1 + y*w2, where
0323 560 : y1 and y2 are the high and low 28 bits of y respectively.
0323 561 :
54 0C AC 7D 0323 562 :
0323 563 : MOVQ exp(AP), R4 ; R4/R5 <-- y
0327 564 :
0327 565 :
0327 566 : Test for the possibility of overflow in the computation of y*w1.
0327 567 : This will occur if the exponent of y plus the exponent of w1 is greater
0327 568 : than 127.
0327 569 :
52 54 08 07 EF 0327 570 :
52 0080 8F A2 032C 571 EXTZV #7, #8, R4, R2 ; biased exp of y
53 56 08 07 EF 0331 572 SUBW2 #^X80, R2 ; unbiased exp of y
53 0080 8F A2 0336 573 EXTZV #7, #8, R6, R3 ; biased exp of w1
53 53 52 A0 033B 574 SUBW2 #^X80, R3 ; unbiased exp of w1
53 007F 8F B1 033E 575 ADDW2 R2, R3 ; unbiased exp of w1*y
; largest unbiased exp possible is 127
50 54 64 0343 576 CMPW #^X7F, R3
52 54 64 0345 577 BLSS Y TIMES_W1_OVER
53 55 FFFF0FFF 8F CB 0348 578 MULD R4, R0 ; R0/R1 <-- y*w2
54 52 62 034B 579 MOVL R4, R2 ; R2 <-- high longword of y1
54 56 64 0353 580 BICL3 #^XFFFF0FFF, R5, R3 ; R2/R3 <-- y1
54 56 64 0356 581 SUBD R2, R4 ; R4/R5 <-- y2
54 56 64 0359 582 MULD R6, R2 ; R2/R3 <-- y1*w1
54 56 64 035C 583 MULD R6, R4 ; R4/R5 <-- y2*w1
035C 584 :
035C 585 :
035C 586 : The next step in computing 2^[y*log2(x)] is to write y*log2(x) as

```



```

03BB 644 ; y*w1 would have caused a hardware system floating overflow error. If y<0,
03BB 645 ; then we should return a result of 0 since result = 2^(y*(w1+w2)). Note,
03BB 646 ; y can not be zero.
03BB 647 ;
03BB 648 ;
03BB 649 Y_TIMES_W1_OVER:
54 73 03BB 650 TSTD R4 ; if y < 0 no overflow is needed
13 14 03BD 651 BGTR OVER ; overflow
03BF 652 ;
03BF 653 ;
03BF 654 ; Underflow; if user has FU set, signal error. Always return 0.0
03BF 655 ;
03BF 656 ;
03BF 657 UNDER:
OB 04 AD 50 7C 03BF 658 CLRQ R0 ; R0/R1 <-- 0
06 E1 03C1 659 BBC #6, SF$W_SAVE_PSW(FP), 2$ ;
7E 00'8F 9A 03C6 660 ; has user enabled floating underflow?
00000000'GF 01 FB 03C6 661 MOVZBL #MTH$K_FLOUNDMAT, -(SP) ; Put underflow code on stack.
03CA 662 CALLS #1, G^MTH$$SIGNAL ; Convert to MTH$_FLOUNDMAT (32-bit
03D1 663 ; VAX-11 exception code) and
03D1 664 ; signal condition
04 03D1 665 2$: RET ; return
03D2 666 ;
03D2 667 ;
03D2 668 ; Signal floating overflow, return reserved operand, -0.0
03D2 669 ;
03D2 670 ;
03D2 671 OVER:
7E 00'8F 9A 03D2 672 MOVZBL #MTH$K_FLOOVEMAT, -(SP) ; else process for overflow
50 01 0F 79 03D6 673 ASHQ #15, #T, R0 ; Put overflow code on stack
03DA 674 ; R0 = result = reserved operand
03DA 675 ; -0.0. R0 will be copied to
03DA 676 ; signal mechanism vector (CHF$MCH_RO/R1)
03DA 677 ; so can be fixed up by any error
00000000'GF 01 FB 03DA 678 CALLS #1, G^MTH$$SIGNAL ; handler
04 03E1 679 RET ; signal condition
03E2 680 ; return - R0 restored from CHF$MCH_RO/R1
03E2 681 .END

```

OTS
VAX
Ass
The
916
The
380
11
Mac

_S2
148
The
MAC

A1_TABLE	00000080	R	02
A2_TABLE	00000108	R	02
ACMASK	= 000007FC		
BASE	= 00000004		
C	00000198	R	02
C1	000001A0	R	02
C2	000001A8	R	02
COMMON	00000252	R	02
DEFINED	0000026F	R	02
EVAL_LOG2	00000281	R	02
EXCEPTION_1	000003B1	R	02
EXCEPTION_2	000003B7	R	02
EXP	= 0000000C		
EXPLEN	= 00000008		
EXPTAB	000001D8	R	02
FEXP	= 00000008		
GET_BASE	0000024E	R	02
INDEX	00000000	R	02
LOGLEN	= 00000005		
LOGTAB	000001B0	R	02
MTH\$\$SIGNAL	*****	X	00
MTH\$K_FLOOVEMAT	*****	X	00
MTH\$K_FLOUNDMAT	*****	X	00
MTH\$K_UNDEXP	*****	X	00
OTSSPOWDD	0000024C	RG	02
OTSSPOWDR	00000218	RG	02
OTSSPOWRD	00000232	RG	02
OVER	000003D2	R	02
RETURN	000003B0	R	02
SFSW_SAVE_PSW	= 00000004		
TWO_M55	00000190	R	02
UNDEFINED	0000025C	R	02
UNDER	000003BF	R	02
Y_TIMES_W1_OVER	000003BB	R	02

+-----+
! Psect synopsis !
+-----+

PSECT name	Allocation	PSECT No.	Attributes
. ABS .	00000000 (0.)	00 (0.)	NOPIC USR CON ABS LCL NOSHR NOEXE NORD NOWRT NOVEC BYTE
\$AB\$\$	00000000 (0.)	01 (1.)	NOPIC USR CON ABS LCL NOSHR EXE RD WRT NOVEC BYTE
_OTSSCODE	000003E? (994.)	02 (2.)	PIC USR CON REL LCL SHR EXE RD NOWRT NOVEC QUAD

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

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

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

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

↑-----↑
! Macro library statistics !
↑-----↑

Macro library name	Macros defined
-----	-----
_\$255\$DUA28:[SYSLIB]STARLET.MLB;2	4

88 GETS were required to define 4 macros.

There were no errors, warnings or information messages.

MACRO/ENABLE=SUPPRESSION/DISABLE=(GLOBAL,TRACEBACK)/LIS=LIS\$:OTSPOWDD/OBJ=OBJ\$:OTSPOWDD MSRC\$:MTHJACKET/UPDATE=(ENH\$:MTHJACKET)+MSRC

0264 AH-BT13A-SE
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY

The image displays a grid of 100 small terminal window screenshots, arranged in a 10x10 pattern. Each window shows a different system command and its output. The commands are often variations of 'MTHTAN LIS' and 'OTSPWCC LIS'. The outputs consist of various system parameters, status information, and data tables. The text is small and difficult to read in detail, but the overall layout is a dense array of these terminal outputs.