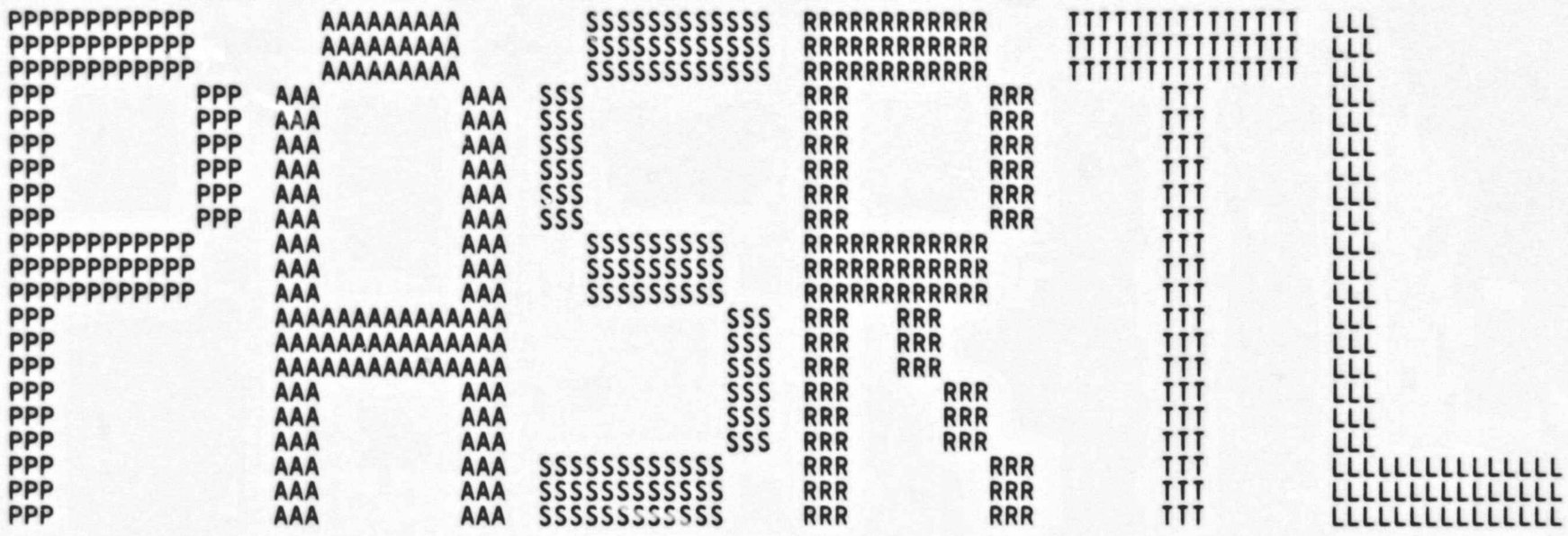
PPPPPPPPPPPP AAAAAAAAA SSSSSSSSSSS RRRRRRRRRRR TTTTTTTTTTTTTTT LLL
PPP PPP AAA AAA SSS RRR RRR TTT LLL
PPPPPPPPPPPP AAAAAAAAAAAAAAA SSS RRR RRR TTT LLL
PPP AAA AAA SSSSSSSSS RRRRRRRRRRR TTT LLL
PPP AAAAAAAAAAAAAAA SSS RRR RRR TTT LLL
PPP AAA AAA SSS RRR RRR TTT LLL
PPP AAA AAA SSSSSSSSSSS RRR RRR TTT LLLLLLLLLLLLLLL

```
PPPPPPPP      AAAAAA      SSSSSSSS    CCCCCCCC  VV      VV  TTTTTTTTTT  RRRRRRRR    TTTTTTTTTT
PPPPPPPP      AAAAAA      SSSSSSSS    CCCCCCCC  VV      VV  TTTTTTTTTT  RRRRRRRR    TTTTTTTTTT
PP      PP   AA      AA  SS          CC        VV      VV      TT      RR      RR      TT
PP      PP   AA      AA  SS          CC        VV      VV      TT      RR      RR      TT
PP      PP   AA      AA  SS          CC        VV      VV      TT      RR      RR      TT
PP      PP   AA      AA  SS          CC        VV      VV      TT      RR      RR      TT
PPPPPPPP     AA      AA    SSSSSS    CC        VV      VV      TT      RRRRRRRR        TT
PPPPPPPP     AA      AA    SSSSSS    CC        VV      VV      TT      RRRRRRRR        TT
PP           AAAAAAAAAA          SS  CC        VV      VV      TT      RR  RR          TT
PP           AAAAAAAAAA          SS  CC        VV      VV      TT      RR  RR          TT
PP           AA      AA          SS  CC          VV  VV        TT      RR    RR        TT          ....
PP           AA      AA          SS  CC          VV  VV        TT      RR    RR        TT          ....
PP           AA      AA  SSSSSSSS    CCCCCCCC      VV          TT      RR      RR      TT          ....
PP           AA      AA  SSSSSSSS    CCCCCCCC      VV          TT      RR      RR      TT          ....

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  PAS$CVTRT - Convert real to text
0000     2          .IDENT  /1-004/                 ; File: PASCVTRT.MAR  Edit: SBL1004
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: Pascal Language Support Library
0000    31 ;
0000    32 ; ABSTRACT:
0000    33 ;
0000    34 ;       This module contains procedures to convert a floating point value
0000    35 ;       to a text representation using either exponential or fixed-point
0000    36 ;       notation.
0000    37 ;
0000    38 ; ENVIRONMENT: User Mode, AST Reentrant
0000    39 ;
0000    40 ;--
0000    41 ; AUTHOR: Steven B. Lionel, CREATION DATE: 12-March-1982
0000    42 ;
0000    43
0000    44 ; Edit History:
0000    45 ;
0000    46 ; 1-001 - Adapted from FOR$CVTRT Edit 1-014.  SBL 12-March-1982
0000    47 ; 1-002 - Take advantage of new OTS$$CVT_F_T_R8 routine.  SBL 14-Mar-1983
0000    48 ; 1-003 - Whoops.  Forgot to fetch address of value for FIXED_POINT in 1-002.
0000    49 ;         SBL 19-Apr-1983
0000    50 ; 1-004 - Move value address fetch in FIXED_POINT.  SBL 18-May-1983
```

```
                     0000     52          .SBTTL  DECLARATIONS
                     0000     53 ;
                     0000     54 ; INCLUDE FILES:
                     0000     55 ;
                     0000     56 ;       NONE
                     0000     57 ;
                     0000     58 ; EXTERNAL DECLARATIONS:
                     0000     59 ;
                     0000     60          .DSABL  GBL                             ; Prevent undeclared
                     0000     61                                                  ; symbols from being
                     0000     62                                                  ; automatically global.
                     0000     63          .EXTRN  OTS$$CVT_D_T_R8                 ; Kernel convert routine
                     0000     64          .EXTRN  OTS$$CVT_F_T_R8                 ; For F floating
                     0000     65          .EXTRN  OTS$$CVT_G_T_R8                 ; For G floating
                     0000     66          .EXTRN  OTS$$CVT_H_T_R8                 ; For H floating
                     0000     67
                     0000     68 ;
                     0000     69 ; MACROS:
                     0000     70 ;
                     0000     71 ;       NONE
                     0000     72 ;
                     0000     73 ; EQUATED SYMBOLS:
                     0000     74 ;
000001FC             0000     75          REGMASK = ^M<R2, R3, R4, R5, R6, R7, R8>
                     0000     76
                     0000     77 ; Stack frame offsets from FP
                     0000     78 ;; Common frame for kernel convert routines
FFFFFFF8             0000     79          PACKED = -8                             ; Temp for packed representation
FFFFFFF4             0000     80          FLAGS = PACKED - 4                      ; Flags for outer and inner routines
FFFFFFF0             0000     81          SIG_DIGITS = FLAGS - 4                  ; Significant digits
FFFFFFEC             0000     82          STRING_ADDR = SIG_DIGITS - 4            ; Address of temp string
FFFFFFE8             0000     83          SIGN = STRING_ADDR - 4                  ; Sign
FFFFFFE4             0000     84          DEC_EXP = SIGN - 4                      ; Decimal exponent
FFFFFFE0             0000     85          OFFSET = DEC_EXP - 4                    ; Offset
FFFFFFDC             0000     86          RT_RND = OFFSET - 4                     ; Right round point
FFFFFFDC             0000     87          COMMON_FRAME = RT_RND                   ; Common frame size
                     0000     88 ;; Not-in-common stack frame
FFFFFFD8             0000     89          EXP_LETTER = COMMON_FRAME - 4           ; Exponent letter to use
FFFFFFD4             0000     90          S_DI = EXP_LETTER - 4                   ; Saved digits in integer
FFFFFFD0             0000     91          S_DE = S_DI - 4                         ; Saved digits in exponent
FFFFFFCC             0000     92          S_DF = S_DE - 4                         ; Saved digits in fraction
FFFFFFC8             0000     93          LEAD_DIGITS = S_DF - 4                  ; Number of leading digits
FFFFFFC4             0000     94          LEAD_ZERO = LEAD_DIGITS - 4             ; Number of zeroes after decimal pt.
FFFFFFC0             0000     95          TRAIL_DIGITS = LEAD_ZERO - 4            ; Number of trailing digits
FFFFFFC0             0000     96          FRAME = TRAIL_DIGITS                    ; Frame size
                     0000     97
02000000             0000     98          M_RT_ROUND = 1@25                       ; Flag to kernel routine
                     0000     99
                     0000    100
                     0000    101 ;
                     0000    102 ; PSECT DECLARATIONS:
                     0000    103 ;
00000000             0000    104          .PSECT _PAS$CODE PIC, USR, CON, REL, LCL, SHR, -
                     0000    105                         EXE, RD, NOWRT, LONG
                     0000    106 ;
                     0000    107 ; OWN STORAGE:
                     0000    108 ;
```

```
                                0000   109
                                0000   110 ;+
                                0000   111 ; Define datatype codes and tables indexed by those codes.
                                0000   112 ;-
                                0000   113
                    00000000    0000   114         DTP_F = 0
                    00000001    0000   115         DTP_D = 1
                    00000002    0000   116         DTP_G = 2
                    00000003    0000   117         DTP_H = 3
                                0000   118
                                0000   119 ;+
                                0000   120 ; Table giving number of exponent digits in exponential format.
                                0000   121 ;-
                                0000   122
                                0000   123 EXP_DIG_TAB:
                    04 03 02 02 0000   124         .BYTE   2,2,3,4
                                0004   125
                                0004   126 ;+
                                0004   127 ; Table giving bit position of exponent.
                                0004   128 ;-
                                0004   129
                                0004   130 POS_TAB:
00000000 00000004 00000007 00000007 0004   131         .LONG   7,7,4,0
                                0014   132
                                0014   133 ;+
                                0014   134 ; Table giving size of exponent in bits.
                                0014   135 ;-
                                0014   136
                                0014   137 SIZE_TAB:
                    0F 0B 08 08 0014   138         .BYTE   8,8,11,15
                                0018   139
                                0018   140 ;+
                                0018   141 ; Table giving exponent biases.  The table values are actually three
                                0018   142 ; lower than the actual bias so that when the unbiased exponent is divided
                                0018   143 ; by three (for fixes-point conversion), the result is what the true result
                                0018   144 ; would be rounded up to the next higher value.
                                0018   145 ;-
                                0018   146
                                0018   147 BIAS_TAB:
00003FFD 000003FD 0000007D 0000007D 0018   148         .LONG   125,125,1021,16381
```

```
                        0028   150          .SBTTL  PAS$CVT_z_T - Convert real to text
                        0028   151 ;++
                        0028   152 ; FUNCTIONAL DESCRIPTION:
                        0028   153 ;
                        0028   154 ;       These procedures convert a floating point value to a text
                        0028   155 ;       representation, and store that representation in a result string.
                        0028   156 ;       The representation is exponential format if the frac_digits
                        0028   157 ;       argument is omitted, and fixed-point notation if it is present.
                        0028   158 ;
                        0028   159 ;       The minimum width of the string written is indicated by the
                        0028   160 ;       contents of the argument actual_width.  This argument is
                        0028   161 ;       modified by the procedure to contain the actual number of
                        0028   162 ;       characters used, which may be more than the minimum width.
                        0028   163 ;
                        0028   164 ;       If the width actually used is less than the minimum width,
                        0028   165 ;       leading blanks are stored.  If the maximum_width is less than
                        0028   166 ;       the width needed, a failure status is returned and the width
                        0028   167 ;       that would be sufficient is stored in actual_width.
                        0028   168 ;
                        0028   169 ;       The syntax of the text representation conforms to that specified
                        0028   170 ;       by VAX-11 Pascal for textfile output of real data.
                        0028   171 ;
                        0028   172 ; CALLING SEQUENCE:
                        0028   173 ;
                        0028   174 ;       status.wlc.v = PAS$CVT_z_T (
                        0028   175 ;                                   value.rz.r,
                        0028   176 ;                                   dest.wt.r,
                        0028   177 ;                                   actual_width.ml.r,
                        0028   178 ;                                   max_width.rl.v
                        0028   179 ;                                   ,[frac_digits.rl.v])
                        0028   180 ;
                        0028   181 ;   where "z" is the datatype (F, D, G or H)
                        0028   182 ;
                        0028   183 ; FORMAL PARAMETERS:
                        0028   184 ;
00000004                0028   185          value           = 4     ; Value to be converted
00000008                0028   186          dest            = 8     ; Destination string
0000000C                0028   187          actual_width    = 12    ; As input, the minimum width of the
                        0028   188                                  ; destination.  As output, the width actually
                        0028   189                                  ; used (or needed in case of error)
00000010                0028   190          max_width       = 16    ; Maximum destination width
00000014                0028   191          frac_digits     = 20    ; Number of fraction digits.  If omitted,
                        0028   192                                  ; then the result is in exponential notation.
                        0028   193                                  ; Otherwise it is in fixed-point notation with
                        0028   194                                  ; the given number of fraction digits.
                        0028   195 ;
                        0028   196 ; IMPLICIT INPUTS:
                        0028   197 ;
                        0028   198 ;       NONE
                        0028   199 ;
                        0028   200 ; IMPLICIT OUTPUTS:
                        0028   201 ;
                        0028   202 ;       NONE
                        0028   203 ;
                        0028   204 ; COMPLETION CODES:
                        0028   205 ;
                        0028   206 ;       1 - Success
```

```
                                0028   207 ;       0 - Failure - The value could not be represented in "max_width"
                                0028   208 ;           characters.  "actual_width" gives the number of characters
                                0028   209 ;           needed for the entire value.
                                0028   210 ;
                                0028   211 ; SIDE EFFECTS:
                                0028   212 ;
                                0028   213 ;       SS$_ROPRAND if the value is a reserved operand.
                                0028   214 ;
                                0028   215 ;--
                                0028   216
                          01FC  0028   217           .ENTRY  PAS$CVT_F_T, REGMASK
58  00000000'GF            9E  002A   218           MOVAB   G^OTS$$CVT_F_T_R8, R8   ; Convert routine address
    57                  00  D0  0031   219           MOVL    #DTP_F, R7              ; Set datatype code
                        28  11  0034   220           BRB     COMMON
                            0036   221
                          01FC  0036   222           .ENTRY  PAS$CVT_D_T, REGMASK
58  00000000'GF            9E  0038   223           MOVAB   G^OTS$$CVT_D_T_R8, R8   ; Convert routine address
    57                  01  D0  003F   224           MOVL    #DTP_D, R7              ; Set datatype code
                        1A  11  0042   225           BRB     COMMON
                            0044   226
                          01FC  0044   227           .ENTRY  PAS$CVT_G_T, REGMASK
58  00000000'GF            9E  0046   228           MOVAB   G^OTS$$CVT_G_T_R8, R8   ; Convert routine address
    57                  02  D0  004D   229           MOVL    #DTP_G, R7              ; Set datatype code
                        0C  11  0050   230           BRB     COMMON
                            0052   231
                          01FC  0052   232           .ENTRY  PAS$CVT_H_T, REGMASK
58  00000000'GF            9E  0054   233           MOVAB   G^OTS$$CVT_H_T_R8, R8   ; Convert routine address
    57                  03  D0  005B   234           MOVL    #DTP_H, R7              ; Set datatype code
                            005E   235 ;             BRB     COMMON
                            005E   236
                            005E   237
                            005E   238 COMMON:
5E  FFFFFFC0 8F             C0  005E   239           ADDL2   #FRAME, SP              ; Create stack frame
        05   6C             91  0065   240           CMPB    (AP), #<frac_digits/4>  ; Frac_digits argument present?
                        03  1F  0068   241           BLSSU   10$                     ; If not, do exponential format
                    008F    31  006A   242           BRW     FIXED_POINT             ; Do fixed-point format
                            006D   243
          F4 AD             D4  006D   244 10$:       CLRL    FLAGS(FP)               ; Clear flags
DO AD   8C AF47             9A  0070   245           MOVZBL  EXP_DIG_TAB[R7], S_DE(FP)  ; Get number of exponent digits
                            0076   246
                            0076   247 ;+
                            0076   248 ; Determine the minimum width and increase actual_width to that if it is smaller.
                            0076   249 ; This is done by the rules of the Pascal standard.
                            0076   250 ;-
                            0076   251
51    D0 AD         06      C1  0076   252           ADDL3   #6, S_DE(FP), R1        ; ActWidth := ExpDigits + 6
      0C BC         51      D1  007B   253           CMPL    R1, @actual_width(AP)   ; Compare with caller width
                    04      15  007F   254           BLEQ    20$                     ; Skip if less than or equal
      0C BC         51      D0  0081   255           MOVL    R1, @actual_width(AP)   ; Store increased width
                            0085   256
                            0085   257 ;+
                            0085   258 ; Check for maximum width exceeded.
                            0085   259 ; Compute number of fraction digits which can be represented in this width,
                            0085   260 ; and get the number of significant digits.  Allocate the kernel convert
                            0085   261 ; routine's temporary string space and call it to do the convert.
                            0085   262 ;-
                            0085   263
```

```
                10 AC    0C BC    D1   0085   264 20$:     CMPL    @actual_width(AP), max_width(AP) ; Not enough characters?
                                   6D   14   008A   265          BGTR    ERROR_E                          ; Error if so
                                   51   D7   008C   266          DECL    R1                               ; Compute fraction digits
             CC AD    0C BC       51   C3   008E   267          SUBL3   R1, @actual_width(AP), S_DF(FP) ; DecPlaces := ActWidth - ExpDigits
             F0 AD    CC AD       01   C1   0094   268          ADDL3   #1, S_DF(FP), SIG_DIGITS(FP) ; Get number of significant digits
                52    F0 AD       13   C1   009A   269          ADDL3   #19, SIG_DIGITS(FP), R2 ; Find temp_string length
                                 5E   52   C2   009F   270          SUBL2   R2, SP                           ; Create string on stack
                      EC AD       5E   D0   00A2   271          MOVL    SP, STRING_ADDR(FP)              ; Temp string address
                                 51   5D   D0   00A6   272          MOVL    FP, R1                           ; Local frame address
                      50    04 AC       D0   00A9   273          MOVL    value(AP), R0                    ; Value address
                                   68   16   00AD   274          JSB     (R8)                             ; Call kernel conversion routine
                                        00AF   275
                                        00AF   276 ;+
                                        00AF   277 ; Determine how many digits are in each field of the output string.
                                        00AF   278 ;-
                                        00AF   279
                   EC AD    E0 AD       C0   00AF   280          ADDL2   OFFSET(FP), STRING_ADDR(FP) ; Get first character pos.
                            E4 AD       D7   00B4   281          DECL    DEC_EXP(FP)                      ; Adjust for leading digit
                            E8 AD       D5   00B7   282          TSTL    SIGN(FP)                         ; Is value zero?
                                        03   12   00BA   283          BNEQ    30$                              ; No
                            E4 AD       D4   00BC   284          CLRL    DEC_EXP(FP)                      ; Yes, exponent is zero
                   C0 AD    CC AD       D0   00BF   285 30$:     MOVL    S_DF(FP), TRAIL_DIGITS(FP)
                      C8 AD    01       D0   00C4   286          MOVL    #1, LEAD_DIGITS(FP)              ; Number of leading digits
                            C4 AD       D4   00C8   287          CLRL    LEAD_ZERO(FP)                    ; No leading zeroes
                      51    0C BC       D0   00CB   288          MOVL    @actual_width(AP), R1            ; Get minimum field width
                   50 D0 AD    04       C1   00CF   289          ADDL3   #4, S_DE(FP), R0                 ; Get width of value
                      50    CC AD       C0   00D4   290          ADDL2   S_DF(FP), R0                     ; Add in fraction digits
                            E8 AD       D5   00D8   291          TSTL    SIGN(FP)                         ; Is value negative?
                                        02   18   00DB   292          BGEQ    40$                              ; Skip if not
                                        50   D6   00DD   293          INCL    R0                               ; Cause one less space to be output
                                        00DF   294
                                        00DF   295 ;+
                                        00DF   296 ; Output the digits and exponent.
                                        00DF   297 ;-
                                        00DF   298
                         00CF       30   00DF   299 40$:     BSBW    DIGITS_OUT                       ; Output digits
                      83    45 8F       90   00E2   300          MOVB    #^A/E/, (R3)+                    ; Move exponent letter
                         54    53       D0   00E6   301          MOVL    R3, R4                           ; Save pointer to exponent field
                F8 AD    05    E4 AD     F9   00E9   302          CVTLP   DEC_EXP(FP), #5, PACKED(FP) ; Convert exponent
             64 D0 AD    F8 AD    05     08   00EF   303          CVTPS   #5, PACKED(FP), S_DE(FP), (R4)
                                   50   D6   00F6   304          INCL    R0                               ; R0 was zeroed by CVTPS, make it 1
                                        04   00F8   305          RET                                      ; Return success
                                        00F9   306
                                        00F9   307 ;+
                                        00F9   308 ; Branch to ERROR_E if the minimum necessary width is wider than the
                                        00F9   309 ; maximum.  The necessary width is already stored in actual_width.
                                        00F9   310 ;-
                                        00F9   311
                                        00F9   312 ERROR_E:
                                   50   D4   00F9   313          CLRL    R0                               ; Indicate failure
                                        04   00FB   314          RET                                      ; return to caller
```

```
                                00FC   316          .SBTTL  FIXED_POINT - Fixed point format
                                00FC   317
                                00FC   318 FIXED_POINT:
                                00FC   319
                                00FC   320 ;+
                                00FC   321 ; Estimate (liberally) how many significant digits we need.
                                00FC   322 ; This is done by getting the unbiased exponent and dividing it
                                00FC   323 ; by 3, which is slightly smaller than log2(10).  This will give
                                00FC   324 ; us a value, perhaps slightly large (which is harmless), for the
                                00FC   325 ; number of digits to the left of the decimal point.  Then add the
                                00FC   326 ; number of fraction digits.
                                00FC   327 ;
                                00FC   328 ; Note:  The bias value in BIAS_TAB is actually three smaller than the
                                00FC   329 ; true exponent bias.  This is so that the number of digits we need is
                                00FC   330 ; rounded up to the next higher number.
                                00FC   331 ;-
                                00FC   332
                     50   04 AC   D0   00FC   333          MOVL    value(AP), R0                   ; Get value address
   60   FF0B CF47    FEFF CF47   EF   0100   334          EXTZV   POS_TAB[R7], SIZE_TAB[R7], (R0), R1 ; Extract exponent
                             51          010A
                  51   FF08 CF47   C2   010B   335          SUBL2   BIAS_TAB[R7], R1                ; Unbias exponent
                       51   03     C6   0111   336          DIVL2   #3, R1                          ; Get power of 10 (approximately)
                            03     18   0114   337          BGEQ    10$                             ; Skip if positive
                       51   01     D0   0116   338          MOVL    #1, R1                          ; Get one digit
                                          0119   339
            F0 AD    14 AC   51   C1   0119   340 10$:     ADDL3   R1, frac_digits(AP), SIG_DIGITS(FP) ; Number of digits needed
                                          011F   341
                                          011F   342 ;+
                                          011F   343 ; Allocate the kernel convert routine's temporary string, specify the
                                          011F   344 ; rounding position, and do the conversion.
                                          011F   345 ;-
                                          011F   346
          52    F0 AD    13   C1   011F   347          ADDL3   #19, SIG_DIGITS(FP), R2 ; Calculate temp string length
                   5E    52   C2   0124   348          SUBL2   R2, SP                  ; Create string on stack
                EC AD    5E   D0   0127   349          MOVL    SP, STRING_ADDR(FP)     ; String address
    F4 AD    02000000   8F   D0   012B   350          MOVL    #M_RT_ROUND, FLAGS(FP)  ; Flag indicating right round
            DC AD    14 AC   D0   0133   351          MOVL    frac_digits(AP), RT_RND(FP)  ; Rounding position
                   51   5D   D0   0138   352          MOVL    FP, R1                  ; Local frame pointer
                        68   16   013B   353          JSB     (R8)                    ; Do the conversion
                                          013D   354
                                          013D   355 ;+
                                          013D   356 ; Get sizes of the various fields in the result string.
                                          013D   357 ;-
                                          013D   358
            EC AD    E0 AD   C0   013D   359          ADDL2   OFFSET(FP), STRING_ADDR(FP) ; Get first digit pos.
                     E8 AD   D5   0142   360          TSTL    SIGN(FP)                ; Is value zero?
                        03   12   0145   361          BNEQ    30$                     ; If zero
                     E4 AD   D4   0147   362          CLRL    DEC_EXP(FP)             ; Then exponent is zero
            C8 AD    E4 AD   D0   014A   363 30$:     MOVL    DEC_EXP(FP), LEAD_DIGITS(FP)    ; Number of leading digits
                        03   18   014F   364          BGEQ    40$                     ; If greater than 0
                     C8 AD   D4   0151   365          CLRL    LEAD_DIGITS(FP)         ; Else no leading digits
            C4 AD    E4 AD   CE   0154   366 40$:     MNEGL   DEC_EXP(FP), LEAD_ZERO(FP)      ; Number of zeroes after dec pt.
                        03   18   0159   367          BGEQ    50$                     ; If greater than 0
                     C4 AD   D4   015B   368          CLRL    LEAD_ZERO(FP)           ; Else no leading zeroes
    C0 AD   14 AC    C4 AD   C3   015E   369 50$:     SUBL3   LEAD_ZERO(FP), frac_digits(AP), TRAIL_DIGITS(FP)
                        10   18   0165   370          BGEQ    60$                     ; If not positive
                     C0 AD   D4   0167   371          CLRL    TRAIL_DIGITS(FP)        ; Then no trailing digits
```

```
                                                D 3

                C4 AD   14 AC    D0  016A   372          MOVL    frac_digits(AP), LEAD_ZERO(FP)
                        C8 AD    D5  016F   373          TSTL    LEAD_DIGITS(FP)                 ; Any significant digits?
                           03    14  0172   374          BGTR    60$                             ; Yes
                        E8 AD    D4  0174   375          CLRL    SIGN(FP)                        ; No, value is +0
          50    C4 AD   C8 AD    C1  0177   376 60$:     ADDL3   LEAD_DIGITS(FP), LEAD_ZERO(FP), R0      ; Compute characters needed
                50      C0 AD    C0  017D   377          ADDL2   TRAIL_DIGITS(FP), R0
                           50    D6  0181   378          INCL    R0                              ; One for the decimal point
                        E8 AD    D5  0183   379          TSTL    SIGN(FP)                        ; Is it negative?
                           02    18  0186   380          BGEQ    70$                             ; Skip if not
                           50    D6  0188   381          INCL    R0                              ; One for the sign
                        C8 AD    D5  018A   382 70$:     TSTL    LEAD_DIGITS(FP)                 ; Leading zero required?
                           02    14  018D   383          BGTR    80$                             ; No
                           50    D6  018F   384          INCL    R0                              ; One for the zero
                                     0191   385
                                     0191   386 ;+
                                     0191   387 ; Compare necessary width with maximum width.
                                     0191   388 ;-
                                     0191   389
             51    0C BC    D0  0191   390 80$:     MOVL    @actual_width(AP), R1           ; Get actual width
                51   50    D1  0195   391          CMPL    R0, R1                          ; Need to expand field?
                           07    15  0198   392          BLEQ    90$                             ; Skip if no
                51         50    D0  019A   393          MOVL    R0, R1                          ; Expand field
             0C BC         50    D0  019D   394          MOVL    R0, @actual_width(AP)           ; Store expanded field size
             10 AC         51    D1  01A1   395 90$:     CMPL    R1, max_width(AP)               ; Maximum width exceeded?
                           07    14  01A5   396          BGTR    ERROR_F                         ; Error if so
                      0007       30  01A7   397          BSBW    DIGITS_OUT                      ; Format the digits
                50         01    D0  01AA   398          MOVL    #1, R0                          ; Indicate success
                           04    04  01AD   399          RET                                     ; Return to caller
                                     01AE   400
                                     01AE   401 ;+
                                     01AE   402 ; Branch to ERROR if the field would expand past max_width.
                                     01AE   403 ;-
                                     01AE   404 ERROR_F:
          50    D4                   01AE   405          CLRL    R0                              ; Indicate failure
                04                   01B0   406          RET                                     ; Return to caller
```

```
                     01B1   408          .SBTTL  DIGITS_OUT
                     01B1   409 ;+
                     01B1   410 ; Routine to format the digits in the output string.
                     01B1   411 ;
                     01B1   412 ;       On entry, R0 contains the number of characters actually needed
                     01B1   413 ;       by the result string.  R1 contains the minimum field width
                     01B1   414 ;
                     01B1   415 ; The string will be constructed as follows:
                     01B1   416 ;
                     01B1   417 ;       n blanks (n is R1-R0)
                     01B1   418 ;       LEAD_DIGITS digits
                     01B1   419 ;       a decimal point
                     01B1   420 ;       LEAD_ZERO zeroes
                     01B1   421 ;       TRAIL_DIGITS digits
                     01B1   422 ;
                     01B1   423 ;       The sign is inserted where appropriate.  If LEAD_DIGITS is
                     01B1   424 ;       zero, a leading zero is inserted.
                     01B1   425 ;
                     01B1   426 ;       Upon exit, R3 points to one byte past where the last character
                     01B1   427 ;       was written.
                     01B1   428 ;-
                     01B1   429 DIGITS_OUT:
56   EC AD    D0     01B1   430          MOVL    STRING_ADDR(FP), R6             ; Address of first digit
53   08 AC    D0     01B5   431          MOVL    dest(AP), R3                    ; Get destination address
   51   50    C2     01B9   432          SUBL2   R0, R1                          ; Get number of leading
                     01BC   433                                                  ; blanks required
        06    15     01BC   434          BLEQ    20$                             ; No blanks needed?
   83   20    90     01BE   435 10$:     MOVB    #^A/ /, (R3)+                   ; Insert leading blanks
     FA 51    F5     01C1   436          SOBGTR  R1, 10$                         ; Loop till done
     E8 AD    D5     01C4   437 20$:     TSTL    SIGN(FP)                        ; Negative?
        03    18     01C7   438          BGEQ    30$                             ; No
   83   2D    90     01C9   439          MOVB    #^A/-/, (R3)+                   ; Minus sign
50   C8 AD    D0     01CC   440 30$:     MOVL    LEAD_DIGITS(FP), R0             ; Check for leading zero
        05    14     01D0   441          BGTR    40$                             ; Not necessary
   83   30    90     01D2   442          MOVB    #^A/0/, (R3)+                   ; Insert zero
        06    11     01D5   443          BRB     50$                             ; Skip leading digits
   83   86    90     01D7   444 40$:     MOVB    (R6)+, (R3)+                    ; Move a digit
     FA 50    F5     01DA   445          SOBGTR  R0, 40$                         ; Loop till done
   83   2E    90     01DD   446 50$:     MOVB    #^A/./, (R3)+                   ; Move decimal point
50   C4 AD    D0     01E0   447          MOVL    LEAD_ZERO(FP), R0               ; Insert leading zeroes
        06    15     01E4   448          BLEQ    70$                             ; Skip if none
   83   30    90     01E6   449 60$:     MOVB    #^A/0/, (R3)+                   ; Move a zero
     FA 50    F5     01E9   450          SOBGTR  R0, 60$                         ; Loop till done
50   C0 AD    D0     01EC   451 70$:     MOVL    TRAIL_DIGITS(FP), R0            ; Move trailing digits
        06    15     01F0   452          BLEQ    90$                             ; Skip if none
   83   86    90     01F2   453 80$:     MOVB    (R6)+, (R3)+                    ; Move trailing digit
     FA 50    F5     01F5   454          SOBGTR  R0, 80$                         ; Loop till done
              05     01F8   455 90$:     RSB                                     ; Return
                     01F9   456
                     01F9   457
                     01F9   458          .END                                    ; End of module PAS$CVTRT
```

```
ACTUAL_WIDTH    = 0000000C
BIAS_TAB          00000018 R      01
COMMON            0000005E R      01
COMMON_FRAME    = FFFFFFDC
DEC_EXP         = FFFFFFE4
DEST            = 00000008
DIGITS_OUT        000001B1 R      01
DTP_D           = 00000001
DTP_F           = 00000000
DTP_G           = 00000002
DTP_H           = 00000003
ERROR_E           000000F9 R      01
ERROR_F           000001AE R      01
EXP_DIG_TAB       00000000 R      01
EXP_LETTER      = FFFFFFD8
FIXED_POINT       000000FC R      01
FLAGS           = FFFFFFF4
FRAC_DIGITS     = 00000014
FRAME           = FFFFFFC0
LEAD_DIGITS     = FFFFFFC8
LEAD_ZERO       = FFFFFFC4
MAX_WIDTH       = 00000010
M_RT_ROUND      = 02000000
OFFSET          = FFFFFFE0
OTS$$CVT_D_T_R8   ********    X   00
OTS$$CVT_F_T_R8   ********    X   00
OTS$$CVT_G_T_R8   ********    X   00
OTS$$CVT_H_T_R8   ********    X   00
PACKED          = FFFFFFF8
PAS$CVT_D_T       00000036 RG     01
PAS$CVT_F_T       00000028 RG     01
PAS$CVT_G_T       00000044 RG     01
PAS$CVT_H_T       00000052 RG     01
POS_TAB           00000004 R      01
REGMASK         = 000001FC
RT_RND          = FFFFFFDC
SIGN            = FFFFFFE8
SIG_DIGITS      = FFFFFFF0
SIZE_TAB          00000014 R      01
STRING_ADDR     = FFFFFFEC
S_DE            = FFFFFFD0
S_DF            = FFFFFFCC
S_DI            = FFFFFFD4
TRAIL_DIGITS    = FFFFFFC0
VALUE           = 00000004
```

Psect synopsis

PSECT name	Allocation	PSECT No.	Attributes
. ABS .	00000000 (0.)	00 (0.)	NOPIC USR CON ABS LCL NOSHR NOEXE NORD NOWRT NOVEC BYTE
_PAS$CODE	000001F9 (505.)	01 (1.)	PIC USR CON REL LCL SHR EXE RD NOWRT NOVEC LONG

Performance indicators

Phase	Page faults	CPU Time	Elapsed Time
Initialization	10	00:00:00.05	00:00:01.54
Command processing	85	00:00:00.63	00:00:04.64
Pass 1	75	00:00:01.19	00:00:03.87
Symbol table sort	0	00:00:00.04	00:00:00.31
Pass 2	92	00:00:00.82	00:00:03.77
Symbol table output	4	00:00:00.07	00:00:01.52
Psect synopsis output	3	00:00:00.02	00:00:00.02
Cross-reference output	0	00:00:00.00	00:00:00.00
Assembler run totals	271	00:00:02.82	00:00:15.67

The working set limit was 750 pages.
7465 bytes (15 pages) of virtual memory were used to buffer the intermediate code.
There were 10 pages of symbol table space allocated to hold 45 non-local and 21 local symbols.
458 source lines were read in Pass 1, producing 20 object records in Pass 2.
0 pages of virtual memory were used to define 0 macros.

Macro library statistics

Macro library name	Macros defined
_255DUA28:[SYSLIB]STARLET.MLB;2	0

0 GETS were required to define 0 macros.

There were no errors, warnings or information messages.

MACRO/ENABLE=SUPPRESSION/DISABLE=(GLOBAL,TRACEBACK)/LIS=LIS$:PASCVTRT/OBJ=OBJ$:PASCVTRT MSRC$:PASCVTRT/UPDATE=(ENH$:PASCVTRT)

PASFVOUTP
LIS

PASEOLN2
LIS

PASHEAP
LIS

PASHANDLE
LIS

PASFAB
LIS

PASGET
LIS

PASCVTRT
LIS

PASDATE
LIS

PASEOF2
LIS

PASFINDK
LIS

PASFVINPU
LIS

PASEXPO
LIS

PASGOTO
LIS

PASFILEUT
LIS

PASHALT
LIS

PASDELETE
LIS

PASFIND2
LIS