```
EEEEEEEEEEEEEEE  MMM         MMM  UUU         UUU  LLL                 AAAAAAAAA     TTTTTTTTTTTTTTT
EEEEEEEEEEEEEEE  MMM         MMM  UUU         UUU  LLL                 AAAAAAAAA     TTTTTTTTTTTTTTT
EEEEEEEEEEEEEEE  MMM         MMM  UUU         UUU  LLL                 AAAAAAAAA     TTTTTTTTTTTTTTT
EEE              MMMMMM   MMMMMM  UUU         UUU  LLL              AAA         AAA        TTT
EEE              MMMMMM   MMMMMM  UUU         UUU  LLL              AAA         AAA        TTT
EEE              MMMMMM   MMMMMM  UUU         UUU  LLL              AAA         AAA        TTT
EEE              MMM   MMM   MMM  UUU         UUU  LLL              AAA         AAA        TTT
EEE              MMM   MMM   MMM  UUU         UUU  LLL              AAA         AAA        TTT
EEE              MMM   MMM   MMM  UUU         UUU  LLL              AAA         AAA        TTT
EEEEEEEEEEEE     MMM         MMM  UUU         UUU  LLL              AAA         AAA        TTT
EEEEEEEEEEEE     MMM         MMM  UUU         UUU  LLL              AAA         AAA        TTT
EEEEEEEEEEEE     MMM         MMM  UUU         UUU  LLL              AAA         AAA        TTT
EEE              MMM         MMM  UUU         UUU  LLL              AAAAAAAAAAAAAAA        TTT
EEE              MMM         MMM  UUU         UUU  LLL              AAAAAAAAAAAAAAA        TTT
EEE              MMM         MMM  UUU         UUU  LLL              AAAAAAAAAAAAAAA        TTT
EEE              MMM         MMM  UUU         UUU  LLL              AAA         AAA        TTT
EEE              MMM         MMM  UUU         UUU  LLL              AAA         AAA        TTT
EEE              MMM         MMM  UUU         UUU  LLL              AAA         AAA        TTT
EEEEEEEEEEEEEEE  MMM         MMM  UUUUUUUUUUUUUUU  LLLLLLLLLLLLLLL  AAA         AAA        TTT
EEEEEEEEEEEEEEE  MMM         MMM  UUUUUUUUUUUUUUU  LLLLLLLLLLLLLLL  AAA         AAA        TTT
EEEEEEEEEEEEEEE  MMM         MMM  UUUUUUUUUUUUUUU  LLLLLLLLLLLLLLL  AAA         AAA        TTT
```

```
_$2

Sym
---
CMP
DEC
DEC
DEC
DEC

DEC
EXE
EXE
EXE
MMC
MMC
MOV
PRS
SYS
VAX
VAX
VAX
VAX
VAX
VAX
VAX
VAX
VAX
VAX
VAX
VAX
VAX
VAX
VAX
VAX
VAX
VAX
VAX
VAX
VAX
VAX
VAX
VAX
VAX
VAX
VAX
VAX
VAX
VAX
VAX
VAX
VAX
VAX
```

```
VV      VV    AAAAAA    XX      XX    CCCCCCCC  VV      VV  TTTTTTTTTT  LL          PPPPPPPP
VV      VV    AAAAAA    XX      XX    CCCCCCCC  VV      VV  TTTTTTTTTT  LL          PPPPPPPP
VV      VV  AA      AA  XX      XX  CC          VV      VV      TT      LL          PP      PP
VV      VV  AA      AA  XX      XX  CC          VV      VV      TT      LL          PP      PP
VV      VV  AA      AA    XX  XX    CC          VV      VV      TT      LL          PP      PP
VV      VV  AA      AA    XX  XX    CC          VV      VV      TT      LL          PP      PP
VV      VV  AA      AA      XX      CC          VV      VV      TT      LL          PPPPPPPP
VV      VV  AA      AA      XX      CC          VV      VV      TT      LL          PPPPPPPP
VV      VV  AAAAAAAAAA    XX  XX    CC          VV      VV      TT      LL          PP
VV      VV  AAAAAAAAAA    XX  XX    CC          VV      VV      TT      LL          PP
  VV  VV    AA      AA  XX      XX  CC            VV  VV        TT      LL          PP              ....
  VV  VV    AA      AA  XX      XX  CC            VV  VV        TT      LL          PP              ....
    VV      AA      AA  XX      XX    CCCCCCCC      VV          TT      LLLLLLLLLL  PP              ....
    VV      AA      AA  XX      XX    CCCCCCCC      VV          TT      LLLLLLLLLL  PP              ....

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          .TITLE  VAX$CVTLP - VAX-11 Instruction Emulator for CVTLP
0000          .IDENT  /V04-000/
0000
000C    ;
0000    ;****************************************************************************
0000    ;*                                                                          *
0000    ;*  COPYRIGHT (c) 1978, 1980, 1982, 1984 BY                                 *
0000    ;*  DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.                  *
0000    ;*  ALL RIGHTS RESERVED.                                                    *
0000    ;*                                                                          *
0000    ;*  THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED   *
0000    ;*  ONLY IN  ACCORDANCE WITH  THE  TERMS  OF  SUCH  LICENSE  AND WITH THE   *
0000    ;*  INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR  ANY  OTHER   *
0000    ;*  COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY   *
0000    ;*  OTHER PERSON.  NO TITLE TO AND OWNERSHIP OF  THE  SOFTWARE IS  HEREBY   *
0000    ;*  TRANSFERRED.                                                            *
0000    ;*                                                                          *
0000    ;*  THE INFORMATION IN THIS SOFTWARE IS  SUBJECT TO CHANGE WITHOUT NOTICE   *
0000    ;*  AND  SHOULD  NOT  BE  CONSTRUED AS  A COMMITMENT BY DIGITAL EQUIPMENT   *
0000    ;*  CORPORATION.                                                            *
0000    ;*                                                                          *
0000    ;*  DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE  OR  RELIABILITY OF ITS   *
0000    ;*  SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.                 *
0000    ;*                                                                          *
0000    ;*                                                                          *
0000    ;****************************************************************************
0000    ;
0000
0000    ;++
0000    ; Facility:
0000    ;
0000    ;       VAX-11 Instruction Emulator
0000    ;
0000    ; Abstract:
0000    ;       The routine in this module emulates the VAX-11 packed decimal
0000    ;       CVTLP instruction. This procedure can be a part of an emulator
0000    ;       package or can be called directly after the input parameters
0000    ;       have been loaded into the architectural registers.
0000    ;
0000    ;       The input parameters to this routine are the registers that
0000    ;       contain the intermediate instruction state.
0000    ;
0000    ; Environment:
0000    ;
0000    ;       This routine runs at any access mode, at any IPL, and is AST
0000    ;       reentrant.
0000    ;
0000    ; Author:
0000    ;
0000    ;       Lawrence J. Kenah
0000    ;
0000    ; Creation Date:
0000    ;
0000    ;       18 October 1983
0000    ;
0000    ; Modified by:
0000    ;
```

```
0000      ;       V01-004 LJK0040         Lawrence J. Kenah       24-Jul-1984
0000      ;               Do not use an INCL instruction to modify the contents of
0000      ;               the sign byte of the output string.
0000      ;
0000      ;       V01-003 LJK0032         Lawrence J. Kenah       5-Jul-1984
0000      ;               Fix restart routine to take into account the fact that restart
0000      ;               codes are based at one when computing restart PC.
0000      ;
0000      ;       V01-002 LJK0024         Lawrence J. Kenah       22-Feb-1984
0000      ;               Add code to handle access violations. Perform minor cleanup.
0000      ;
0000      ;       V01-001 LJK0008         Lawrence J. Kenah       18-Oct-1983
0000      ;               The emulation code for CVTLP was moved into a separate module.
0000      ;--
```

```
    0000          .SUBTITLE       Declarations
    0000
    0000  ; Include files:
    0000
    0000          .NOCROSS                        ; No cross reference for these
    0000          .ENABLE         SUPPRESSION     ; No symbol table entries either
    0000
    0000          CVTLP_DEF                       ; Bit fields in CVTLP registers
    0000          PACK_DEF                        ; Stack usage by exception handler
    0000          STACK_DEF                       ; Stack usage for original exception
    0000
    0000          $PSLDEF                         ; Define bit fields in PSL
    0000
    0000          .DISABLE        SUPPRESSION     ; Turn on symbol table again
    0000          .CROSS                          ; Cross reference is OK now
    0000
    0000  ; External declarations
    0000
    0000          .DISABLE        GLOBAL
    0000
    0000          .EXTERNAL -
    0000                          DECIMAL$BINARY_TO_PACKED_TABLE
    0000
    0000          .EXTERNAL -
    0000                          VAX$EXIT_EMULATOR,-
    0000                          VAX$REFLECT_FAULT,-
    0000                          VAX$ROPRAND,-
    0000                          VAX$DECIMAL_OVERFLOW
    0000
    0000  ; PSECT Declarations:
    0000
    0000          .DEFAULT        DISPLACEMENT , WORD
    0000
00000000          .PSECT _VAX$CODE PIC, USR, CON, REL, LCL, SHR, EXE, RD, NOWRT, LONG
    0000
    0000          BEGIN_MARK_POINT        RESTART
```

```
              0000          .SUBTITLE       VAX$CVTLP - Convert Long to Packed
              0000  ;+
              0000  ; Functional Description:
              0000  ;
              0000  ;       The source operand is converted to  a  packed  decimal  string  and  the
              0000  ;       destination  string  operand  specified  by  the  destination length and
              0000  ;       destination address operands is replaced by the result.
              0000  ;
              0000  ; Input Parameters:
              0000  ;
              0000  ;       R0 - src.rl             Input longword to be converted
              0000  ;       R2 - dstlen.rw          Length of output decimal string
              0000  ;       R3 - dstaddr.ab         Address of output packed decimal string
              0000  ;
              0000  ; Output Parameters:
              0000  ;
              0000  ;       R0 = 0
              0000  ;       R1 = 0
              0000  ;       R2 = 0
              0000  ;       R3 = Address of byte containing most significant digit of
              0000  ;            the destination string
              0000  ;
              0000  ; Condition Codes:
              0000  ;
              0000  ;       N <- destination string LSS 0
              0000  ;       Z <- destination string EQL 0
              0000  ;       V <- decimal overflow
              0000  ;       C <- 0
              0000  ;
              0000  ; Register Usage:
              0000  ;
              0000  ;       This routine uses R0 through R5 and R11 as scratch registers. R10
              0000  ;       serves its usual function as an access violation routine pointer. The
              0000  ;       condition codes are stored in R11 as the routine executes.
              0000  ;
              0000  ; Notes:
              0000  ;
              0000  ;       The algorithm used in this routine builds the packed decimal from
              0000  ;       least significant digit to most significant digit. The least
              0000  ;       significant digit is obtained by dividing the input longword by 10 and
              0000  ;       storing the remainder as the least significant digit. The rest of the
              0000  ;       result is obtained by taking the quotient from the first step,
              0000  ;       repeatedly dividing by 100, and converting the resulting remainder
              0000  ;       into a pair of packed decimal digits. This process continues until the
              0000  ;       quotient goes to zero.
              0000  ;
              0000  ;       No special processing is observed for an input longword of zero. The
              0000  ;       correct results for this case drops out of normal processing.
              0000  ;-
              0000
              0000          .ENABLE         LOCAL_BLOCK
              0000
              0000          ASSUME CVTLP_B_STATE EQ 7       ; Make sure we test the right FPD bit
              0000
 0142    31   0000  2$:     BRW     VAX$CVTLP_RESTART       ; Restart somewhere else
              0003
              0003  VAX$CVTLP::
```

```
           F9 51    1B   E0  0003          BBS     #<CVTLP_V_FPD+24>,R1,2$ ; Branch if this is a restart
              0C30 8F    BB  0007          PUSHR   #^M<R4,R5,R10,R11>      ; Save some registers
                             000B          ESTABLISH_HANDLER      -        ; Store address of access
                             000B                  CVTLP_ACCVIO            ;  violation handler
                             0010
                             0010     ; Get initial settings for condition codes. The initial settings for V and C
                             0010     ; will be zero. The initial setting of N depends on the sign of the source
                             0010     ; operand. The Z-bit starts off set and remains set until a nonzero digit is
                             0010     ; stored in the output string. Note that the final Z-bit may be set for
                             0010     ; nonzero input if the output string is not large enough. (The V-bit is set
                             0010     ; in this case.) In this case, the saved DV bit will determine whether to
                             0010     ; reflect an exception or merely report the result to the caller.
                             0010
                        5B   DC  0010          MOVPSL  R11                     ; Get DV bit from PSL on input
      5B    04    00    04   F0  0012          INSV    #PSL$M_Z,#0,#4,R11      ; Start with Z-bit set, others clear
                                 0017          ROPRAND_CHECK   R2              ; Insure that R2 LEQU 31
        51    52    FF  8F   78  0022          ASHL    #-1,R2,R1               ; Convert digit count to byte count
                  53    51   C0  0027          ADDL    R1,R3                   ; Get address of sign byte
                                 002A          MARK_POINT      CVTLP_1 , RESTART
                  63    0C   90  002A          MOVB    #12,(R3)                ; Assume that sign is PLUS
                        50   D5  002D          TSTL    R0                      ; Check sign of source operand
                        08   18  002F          BGEQ    10$                     ; Start getting digits if not negative
                                 0031
                                 0031     ; Source operand is minus. We remember that by setting the saved N-bit but work
                                 0031     ; with the absolute value of the input operand from this point on.
                                 0031
                                 0031          MARK_POINT      CVTLP_2 , RESTART
                        63   96  0031          INCB    (R3)                    ; Convert "+" to "-" (12 -> 13)
                  50    50   CE  0033          MNEGL   R0,R0                   ; Normalize source operand
                  5B    08   88  0036          BISB    #PSL$M_N,R11            ; Set N-bit in saved PSW
                                 0039
                                 0039     ;+
                                 0039     ; The first (least significant) digit is obtained by dividing the source
                                 0039     ; longword by ten and storing the remainder in the high order nibble of the
                                 0039     ; sign byte. Note that at this point, the upper four bits of the sign byte
                                 0039     ; contain zero.
                                 0039     ;-
                                 0039
                        51   D4  0039 10$:     CLRL    R1                      ; Prepare R1 for input to EDIV
                  54    52   D0  003B          MOVL    R2,R4                   ; Special exit if zero source length
                        64   13  003E          BEQL    90$                     ; Only overflow check remains
      55    50    50    0A   7B  0040          EDIV    #10,R0,R0,R5            ; R5 gets remainder, first digit
            55    55    04   78  0045          ASHL    #4,R5,R5                ; Shift digit to high nibble position
                        06   13  0049          BEQL    20$                     ; Leave Z-bit alone if digit is zero
                  5B    04   8A  004B          BICB    #PSL$M_Z,R11            ; Turn off Z-bit if nonzero
                                 004E          MARK_POINT      CVTLP_3 , RESTART
                  63    55   80  004E          ADDB    R5,(R3)                 ; Merge this digit with low nibble
                        54   D7  0051 20$:     DECL    R4                      ; One less output digit
                        4F   13  0053          BEQL    90$                     ; No more room in output string
        54    54    FF  8F   78  0055          ASHL    #-1,R4,R4               ; Number of complete bytes remaining
                        38   13  005A          BEQL    80$                     ; Check for last digit if none
                        50   D5  005C          TSTL    R0                      ; Is source exhausted?
                        04   12  005E          BNEQ    30$                     ; Go get next digits if not
                                 0060          MARK_POINT      CVTLP_4 , RESTART
                        73   94  0060          CLRB    -(R3)                   ; Store a pair of zeros
                        1D   11  0062          BRB     50$                     ; Fill rest of output with zeros
                                 0064
```

```
                                    0064   ;+
                                    0064   ; The following loop obtains two digits at a time from the source longword. It
                                    0064   ; accomplishes this by dividing the current value of R0 by 100 and converting
                                    0064   ; the remainder to a pair of decimal digits using the table that converts
                                    0064   ; binary numbers in the range from 0 to 99 to their packed decimal equivalents.
                                    0064   ; Note that this technique may cause nonzero to be stored in the upper nibble
                                    0064   ; of the most significant byte of an even length string. This condition will
                                    0064   ; be tested for at the end of the loop.
                                    0064   ;-
                                    0064
55   50   50   00000064 8F     7B   0064   30$:    EDIV    #100,R0,R0,R5           ; R5 gets remainder, next digit
                                    006D           MARK_POINT      CVTLP_5 , RESTART
           73   0000'CF45      90   006D           MOVB    DECIMAL$BINARY_TO_PACKED_TABLE[R5],-(R3)
                                    0073                                           ; Store converted remainder
                     03        13   0073           BEQL    40$                     ; Leave Z-bit alone if digit is zero
                5B   04        8A   0075           BICB    #PSL$M_Z,R11            ; Turn off Z-bit if nonzero
                     50        D5   0078   40$:    TSTL    R0                      ; Is source exhausted?
                     05        13   007A           BEQL    50$                     ; Exit loop is no more source
                E5   54        F5   007C           SOBGTR  R4,30$                  ; Check for end of loop
                                    007F
                     13        11   007F           BRB     80$                     ; Check for remaining digit
                                    0081
                                    0081   ; The following code executes if the source longword is exhausted. If there
                                    0081   ; are any remaining digits in the destination string, they must be filled
                                    0081   ; with zeros. Note that one more byte is cleared if the original input length
                                    0081   ; was odd. This includes the most significant digit and the unused nibble.
                                    0081
                02   52        E8   0081   50$:    BLBS    R2,65$                  ; One less byte to zero if odd input length
                                    0084
                                    0084           MARK_POINT      CVTLP_6 , RESTART
                     73        94   0084   60$:    CLRB    -(R3)                   ; Set a pair of digits to zero
                FB   54        F5   0086   65$:    SOBGTR  R4,60$                  ; Any more digits to zero?
                                    0089
                                    0089   ; The following code is the exit path for this routine. Note that all code
                                    0089   ; paths that arrive here do so with R0 containing zero. R1 and R2, however,
                                    0089   ; must be cleared on exit.
                                    0089
                     51        7C   0089   70$:    CLRQ    R1                      ; Conform to architecture
                     0F        B9   008B           BICPSW  #<PSL$M_N!PSL$M_Z!PSL$M_V!PSL$M_C>      ; Clear condition codes
                     5B        B8   008D           BISPSW  R11                     ; Set appropriate condition codes
              0C30   8F        BA   008F           POPR    #^M<R4,R5,R10,R11>      ; Restore registers, preserving PSW
                               05   0093           RSB
                                    0094
                                    0094   ;+
                                    0094   ; The following code executes when there is no more room in the destination
                                    0094   ; string. We first test for the parity of the output length and, if even,
                                    0094   ; determine whether a nonzero digit was stored in the upper nibble of the
                                    0094   ; most significant byte. Such a nonzero store causes an overflow condition.
                                    0094   ;
                                    0094   ; If the source operand is not yet exhausted, then decimal overflow occurs.
                                    0094   ; If decimal overflow exceptions are enabled, an exception is signalled.
                                    0094   ; Otherwise, the V-bit in the PSW is set and a normal exit is issued. Note
                                    0094   ; that negative zero is only an issue for this instruction when overflow
                                    0094   ; occurs. In the no overflow case, the entire converted longword is stored in
                                    0094   ; the output string and there is only one form of binary zero.
                                    0094   ;-
                                    0094
```

```
             0D 52    E8  0094   80$:    BLBS    R2,90$                   ; No last digit if odd output length
  55    50    50 0A   7B  0097           EDIV    #10,R0,R0,R5             ; Get next input digit
                          009C           MARK_POINT       CVTLP_7 , RESTART
              73 55   90  009C           MOVB    R5,-(R3)                 ; Store in last output byte
                 03   13  009F           BEQL    90$                      ; Leave Z-bit alone if zero
              5B 04   8A  00A1           BICB    #PSL$M_Z,R11
                          00A4
                 50   D5  00A4   90$:    TSTL    R0                       ; Is source also all used up?
                 E1   13  00A6           BEQL    70$                      ; Yes, continue with exit processing
                          00A8
                          00A8   ; An overflow has occurred. If the Z-bit is still set, then the N-bit is cleared.
                          00A8   ; Note that, because all negative zero situations occur simultaneously with
                          00A8   ; overflow, the output sign is left as minus.
                          00A8
                 50   D4  00A8   100$:   CLRL    R0                       ; R0 must be zero on exit
           03 5B 02   E1  00AA           BBC     #PSL$V_Z,R11,110$        ; Z-bit and N-bit cannot both be set
              5B 08   8A  00AE           BICB    #PSL$M_N,R11             ; Clear N-bit if Z-bit still set
              5B 02   88  00B1   110$:   BISB    #PSL$M_V,R11             ; Set V-bit in saved PSW
                          00B4
                          00B4   ; If the V-bit is set and decimal traps are enabled (DV-bit is set), then
                          00B4   ; a decimal overflow trap is generated. Note that the DV-bit can be set in
                          00B4   ; the current PSL or, if this routine was entered as the result of an emulated
                          00B4   ; instruction exception, in the saved PSL on the stack.
                          00B4
           10 5B 07   E0  00B4           BBS     #PSL$V_DV,R11,120$       ; Report exception if current DV-bit set
         54   0000'CF 9E  00B8           MOVAB   VAX$EXIT_EMULATOR,R4     ; Set up R4 for PIC address comparison
           10 AE 54   D1  00BD           CMPL    R4,<4*4>(SP)             ; Is return PC EQLU VAX$EXIT_EMULATOR ?
                 C6   12  00C1           BNEQU   70$                      ; No. Simply return V-bit set
        C1 40 AE 07   E1  00C3           BBC     #PSL$V_DV,<<4*<4+1>>+EXCEPTION_PSL>(SP),70$
                          00C8                                            ; Only return V-bit if DV-bit is clear
                          00C8
                          00C8   ; Restore the saved registers and transfer control to DECIMAL_OVERFLOW
                          00C8
                 51   7C  00C8   120$:   CLRQ    R1                       ; Comform to architecture
                 0F   B9  00CA           BICPSW  #<PSL$M_N!PSL$M_Z!PSL$M_V!PSL$M_C>      ; Clear condition codes
                 5B   B8  00CC           BISPSW  R11                      ; Set appropriate condition codes
            0C30 8F   BA  00CE           POPR    #^M<R4,R5,R10,R11>       ; Restore registers, preserving PSW
               FF2B'  31  00D2           BRW     VAX$DECIMAL_OVERFLOW     ; Report overflow exception
                          00D5
                          00D5           .DISABLE        LOCAL_BLOCK
```

```
                     00D5          .SUBTITLE       DECIMAL_ROPRAND
                     00D5   ;-
                     00D5   ; Functional Description:
                     00D5   ;
                     00D5   ;      This routine receives control when a digit count larger than 31
                     00D5   ;      is detected. The exception is architecturally defined as an
                     00D5   ;      abort so there is no need to store intermediate state. The digit
                     00D5   ;      count is made after registers are saved. These registers must be
                     00D5   ;      restored before reporting the exception.
                     00D5   ;
                     00D5   ; Input Parameters:
                     00D5   ;
                     00D5   ;      00(SP) - Saved R4
                     00D5   ;      04(SP) - Saved R5
                     00D5   ;      08(SP) - Saved R10
                     00D5   ;      12(SP) - Saved R11
                     00D5   ;      16(SP) - Return PC from VAX$CVTLP routine
                     00D5   ;
                     00D5   ; Output Parameters:
                     00D5   ;
                     00D5   ;      00(SP) - Offset in packed register array to delta PC byte
                     00D5   ;      04(SP) - Return PC from VAX$CVTLP routine
                     00D5   ;
                     00D5   ; Implicit Output:
                     00D5   ;
                     00D5   ;      This routine passes control to VAX$ROPRAND where further
                     00D5   ;      exception processing takes place.
                     00D5   ;-
                     00D5
                     00D5   DECIMAL_ROPRAND:
0C30 8F      BA      00D5          POPR    #^M<R4,R5,R10,R11>      ; Restore registers
      0B      DD      00D9          PUSHL   #CVTLP_B_DELTA_PC       ; Store offset to delta PC byte
   FF22'      31      00DB          BRW     VAX$ROPRAND             ; Pass control along
```

```
                        00DE         .SUBTITLE       CVTLP_ACCVIO - Reflect an Access Violation
                        00DE    ;+
                        00DE    ; Functional Description:
                        00DE    ;
                        00DE    ;       This routine receives control when an access violation occurs while
                        00DE    ;       executing within the VAX$CVTLP emulator routine.
                        00DE    ;
                        00DE    ;       The routine header for ASHP_ACCVIO in module VAX$ASHP contains a
                        00DE    ;       detailed description of access violation handling for the decimal
                        00DE    ;       string instructions. This routine differs from most decimal
                        00DE    ;       instruction emulation routines in that it preserves intermediate
                        00DE    ;       results if an access violation occurs. This is accomplished by
                        00DE    ;       storing the number of the exception point, as well as intermediate
                        00DE    ;       arithmetic results, in the registers R0 through R3.
                        00DE    ;
                        00DE    ; Input Parameters:
                        00DE    ;
                        00DE    ;       See routine ASHP_ACCVIO in module VAX$ASHP
                        00DE    ;
                        00DE    ; Output Parameters:
                        00DE    ;
                        00DE    ;       See routine ASHP_ACCVIO in module VAX$ASHP
                        00DE    ;-
                        00DE
                        00DE    CVTLP_ACCVIO:
             52    D4   00DE            CLRL    R2                              ; Initialize the counter
        FF1C CF    9F   00E0            PUSHAB  MODULE_BASE                     ; Store base address of this module
          51  8E   C2   00E4            SUBL2   (SP)+,R1                        ; Get PC relative to this base
                        00E7
  0000'CF42  51    B1   00E7    10$:    CMPW    R1,PC_TABLE_BASE[R2]            ; Is this the right PC?
             07    13   00ED            BEQL    30$                             ; Exit loop if true
       F4 52 07    F2   00EF            AOBLSS  #TABLE_SIZE,R2,10$              ; Do the entire table
                        00F3
                        00F3    ; If we drop through the dispatching based on PC, then the exception is not
                        00F3    ; one that we want to back up. We simply reflect the exception to the user.
                        00F3
             0F    BA   00F3    20$:    POPR    #^M<R0,R1,R2,R3>                ; Restore saved registers
                   05   00F5            RSB                                     ; Return to exception dispatcher
                        00F6
                        00F6    ; The exception PC matched one of the entries in our PC table. R2 contains
                        00F6    ; the index into both the PC table and the handler table. R1 has served
                        00F6    ; its purpose and can be used as a scratch register.
                        00F6
  51   0000'CF42   3C   00F6    30$:    MOVZWL  HANDLER_TABLE_BASE[R2],R1       ; Get the offset to the handler
        FEFF CF41  17   00FC            JMP     MODULE_BASE[R1]                 ; Pass control to the handler
                        0101
                        0101    ; In all of the instruction-specific routines, the state of the stack
                        0101    ; will be shown as it was when the exception occurred. All offsets will
                        0101    ; be pictured relative to R0.
```

```
0101            .SUBTITLE       Context-Specific Access Violation Handling for VAX$CVTLP
0101    ;+
0101    ; Functional Description:
0101    ;
0101    ;       The intermediate state of the instruction is packed into registers R0
0101    ;       through R3 and control is passed to VAX$REFLECT_FAULT that will, in
0101    ;       turn, reflect the access violation back to the user. The intermediate
0101    ;       state reflects the point at which the routine was executing when the
0101    ;       access violation occurred.
0101    ;
0101    ; Input Parameters:
0101    ;
0101    ;       R0 - Address of top of stack when access violation occurred
0101    ;
0101    ;       00(R0) - Saved R4 on entry to VAX$CVTLP
0101    ;       04(R0) - Saved R5
0101    ;       08(R0) - Saved R10
0101    ;       12(R0) - Saved R11
0101    ;       16(R0) - Return PC from VAX$CVTLP routine
0101    ;
0101    ;       00(SP) - Saved R0 (restored by VAX$HANDLER)
0101    ;       04(SP) - Saved R1
0101    ;       08(SP) - Saved R2
0101    ;       12(SP) - Saved R3
0101    ;
0101    ; Output Parameters:
0101    ;
0101    ;       R0 - Address of return PC from VAX$CVTLP
0101    ;       R1 - Byte offset to delta-PC in saved register array
0101    ;               (PACK_V_FPD and PACK_M_ACCVIO set to identify exception)
0101    ;
0101    ;       See list of input parameters for CVTLP_RESTART for a description of the
0101    ;       contents of the packed register array.
0101    ;
0101    ; Implicit Output:
0101    ;
0101    ;       R4, R5, R10, and R11 are restored to the values that they had
0101    ;       when VAX$CVTLP was entered.
0101    ;-
0101
0101            .ENABLE         LOCAL_BLOCK
0101
0101    ;+
0101    ; CVTLP_1 or CVTLP_2
0101    ;
0101    ; An access violation occurred while storing the initial sign in the output
0101    ; string. R1, R4, and R5 contain junk at this point.
0101    ;
0101    ;       R0  - Input source longword
0101    ;       R2  - Digit count of destination string
0101    ;       R3  - Address of sign byte in destination string
0101    ;       R11 - Current PSW (with Z-bit set and all others clear)
0101    ;
0101    ;       R1 - Not important
0101    ;       R4 - Scratch but saved anyway
0101    ;       R5 - Scratch but saved anyway
0101    ;-
```

```
                       0101
                       0101  CVTLP_1:
                 90    0101          MOVB    #<CVTLP_1_RESTART!-      ; Store code that locates exception PC
                       0102                  CVTLP_M_FPD>,-
   07 AE   09          0102                  CVTLP_B_STATE(SP)
           22    11    0105          BRB     10$                      ; Join common code
                       0107
                       0107  CVTLP_2:
                 90    0107          MOVB    #<CVTLP_2_RESTART!-      ; Store code that locates exception PC
                       0108                  CVTLP_M_FPD>,-
   07 AE   0A          0108                  CVTLP_B_STATE(SP)
           1C    11    010B          BRB     10$                      ; Join common code
                       010D
                       010D  ;+
                       010D  ; CVTLP_3 through CVTLP_7
                       010D  ;
                       010D  ; An access violation occurred while storing a digit or digit pair in the
                       010D  ; output string.
                       010D  ;
                       010D  ;       R0  - Input source longword (updated)
                       010D  ;       R1  - Zero (so that R0/R1 can be used as input quadword to EDIV)
                       010D  ;       R2  - Digit count of destination string
                       010D  ;       R3  - Address of current byte in destination string
                       010D  ;       R4  - Updated digit or byte count
                       010D  ;       R5  - Most recent remainder from EDIV
                       010D  ;       R11 - Current PSW (condition codes reflect results so far)
                       010D  ;-
                       010D
                       010D  CVTLP_3:
                 90    010D          MOVB    #<CVTLP_3_RESTART!-      ; Store code that locates exception PC
                       010E                  CVTLP_M_FPD>,-
   07 AE   0B          010E                  CVTLP_B_STATE(SP)
           16    11    0111          BRB     10$                      ; Join common code
                       0113
                       0113  CVTLP_4:
                 90    0113          MOVB    #<CVTLP_4_RESTART!-      ; Store code that locates exception PC
                       0114                  CVTLP_M_FPD>,-
   07 AE   0C          0114                  CVTLP_B_STATE(SP)
           10    11    0117          BRB     10$                      ; Join common code
                       0119
                       0119  CVTLP_5:
                 90    0119          MOVB    #<CVTLP_5_RESTART!-      ; Store code that locates exception PC
                       011A                  CVTLP_M_FPD>,-
   07 AE   0D          011A                  CVTLP_B_STATE(SP)
           0A    11    011D          BRB     10$                      ; Join common code
                       011F
                       011F  CVTLP_6:
                 90    011F          MOVB    #<CVTLP_6_RESTART!-      ; Store code that locates exception PC
                       0120                  CVTLP_M_FPD>,-
   07 AE   0E          0120                  CVTLP_B_STATE(SP)
           04    11    0123          BRB     10$                      ; Join common code
                       0125
                       0125  CVTLP_7:
                 90    0125          MOVB    #<CVTLP_7_RESTART!-      ; Store code that locates exception PC
                       0126                  CVTLP_M_FPD>,-
   07 AE   0F          0126                  CVTLP_B_STATE(SP)
                       0129
```

```
          04 AE    54   90  0129  10$:    MOVB    R4,CVTLP_B_SAVED_R4(SP) ; Store current digit/byte count
          05 AE    55   90  012D          MOVB    R5,CVTLP_B_SAVED_R5(SP) ; Store latest EDIV remainder
          06 AE    5B   90  0131          MOVB    R11,CVTLP_B_SAVED_PSW(SP) ; Store current condition codes
                            0135
                            0135  ; At this point, all intermediate state has been preserved in the register
                            0135  ; array on the stack. We now restore the registers that were saved on entry
                            0135  ; to VAX$CVTLP and pass control to VAX$REFLECT_FAULT where further exception
                            0135  ; dispatching takes place.
                            0135
             54    80   7D  0135          MOVQ    (R0)+,R4                ; Restore R4 and R5
             5A    80   7D  0138          MOVQ    (R0)+,R10               ; ... and R10 and R11
                            013B
   51   0000030B 8F     D0  013B          MOVL    #<CVTLP_B_DELTA_PC!-    ; Indicate offset for delta PC
                            0142                  PACK_M_FPD!-            ; FPD bit should be set
                            0142                  PACK_M_ACCVIO>,R1       ; This is an access violation
                FEBB'   31  0142          BRW     VAX$REFLECT_FAULT       ; Continue exception handling
                            0145
                            0145          .DISABLE        LOCAL_BLOCK
```

```
0145            .SUBTITLE       CVTLP_RESTART - Unpack and Restart CVTLP Instruction
0145    ;+
0145    ; Functional Description:
0145    ;
0145    ;       This routine receives control when a CVTLP instruction is restarted.
0145    ;       The instruction state (stack and general registers) is restored to the
0145    ;       state that it was in when the instruction (routine) was interrupted and
0145    ;       control is passed to the PC at which the exception occurred.
0145    ;
0145    ; Input Parameters:
0145    ;
0145    ;       31              23              15              07             00
0145    ;      +----------------+----------------+----------------+----------------+
0145    ;      :                                src                                : : R0
0145    ;      +----------------+----------------+----------------+----------------+
0145    ;      :     state      :   saved_PSW    :    saved_R5    :    saved_R4    : : R1
0145    ;      +----------------+----------------+----------------+----------------+
0145    ;      ,   delta-PC     :      XXXX      :              dstlen             : : R2
0145    ;      +----------------+----------------+----------------+----------------+
0145    ;      :                              dstaddr                              : : R3
0145    ;      +----------------+----------------+----------------+----------------+
0145    ;
0145    ;       Depending on where the exception occurred, some of these parameters
0145    ;       may not be relevant. They are nevertheless stored as if they were
0145    ;       valid to make this restart code as simple as possible.
0145    ;
0145    ;       R0          - Updated source longword
0145    ;       R1<07:00> - Latest digit or byte count (loaded into R4)
0145    ;       R1<15:08> - Most recent remainder from EDIV (loaded into R5)
0145    ;       R1<23:16> - Saved condition codes (loaded into R11)
0145    ;       R1<26:24> - Restart code (identifies point where routine will resume)
0145    ;       R1<27>      - Internal FPD flag
0145    ;       R2<15:00> - Initial value of "dstlen"
0145    ;       R2<23:16> - spare
0145    ;       R2<31:24> - Size of instruction in instruction stream
0145    ;       R3          - Address of current byte in destination string
0145    ;
0145    ;       00(SP) - Return PC from VAX$CVTLP routine
0145    ;
0145    ; Output Parameters:
0145    ;
0145    ;       R0  - Updated source longword (unchanged from input)
0145    ;       R1  - scratch
0145    ;       R2  - Initial value of "dstlen"
0145    ;       R3  - Address of current byte in output string (unchanged from input)
0145    ;       R4  - Latest digit or byte count
0145    ;       R5  - Most recent remainder from EDIV
0145    ;       R10 - Address of CVTLP_ACCVIO, this module's "condition handler"
0145    ;       R11 - Condition codes
0145    ;
0145    ;       00(SP) - Saved R4
0145    ;       04(SP) - Saved R5
0145    ;       08(SP) - Saved R10
0145    ;       12(SP) - Saved R11
0145    ;       16(SP) - Return PC from VAX$CVTLP routine
0145    ;
0145    ; Implicit Output:
```

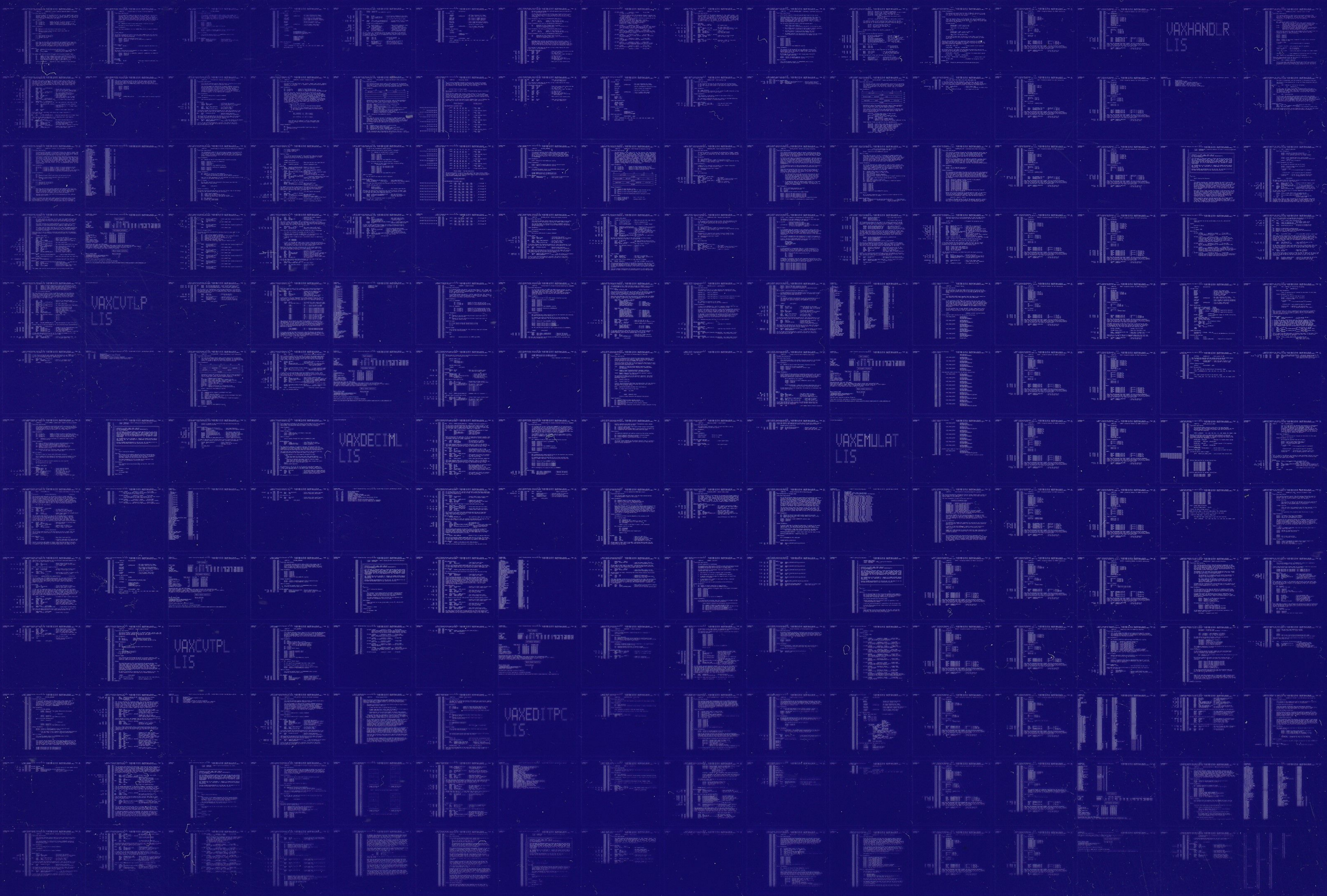
```
                               0145  ;
                               0145  ;         Control is passed to the instruction that was executing when the
                               0145  ;         access violation occurred.
                               0145  ;-
                               0145
                               0145  VAX$CVTLP_RESTART::
            0C33 8F     BB     0145          PUSHR   #^M<R0,R1,R4,R5,R10,R11>          ; Save some registers
                               0149          ESTABLISH_HANDLER       CVTLP_ACCVIO      ; Reload R10 with handler address
                 00     EF     014D          EXTZV   #CVTLP_V_STATE,-
                 03            014F                  #CVTLP_S_STATE,-
         51   07 AE            0150                  CVTLP_B_STATE(SP),R1              ; Put restart code into R1
         54   04 AE     9A     0153          MOVZBL  CVTLP_B_SAVED_R4(SP),R4           ; Restore digit/byte count
         55   05 AE     9A     0157          MOVZBL  CVTLP_B_SAVED_R5(SP),R5           ; Restore latest EDIV remainder
         5B   06 AE     9A     015B          MOVZBL  CVTLP_B_SAVED_PSW(SP),R11         ; Restore condition codes
            52    52    9A     015F          MOVZBL  R2,R2                             ; Clear out R2<31:8>
            5E    08    C0     0162          ADDL    #8,SP                             ; Discard saved R0 and R1
   51    FFFE'CF41      3C     0165          MOVZWL  RESTART_PC_TABLE_BASE-2[R1],R1    ; Convert code to PC offset
                               016B
                               016B  ; In order to get back to the restart point with R1 containing zero, we cannot
                               016B  ; use R1 to transfer control as we did in other routines like VAX$CVTPL.
                               016B
            FE90 CF41   9F     016B          PUSHAB  MODULE_BASE[R1]          ; Store "return PC"
                   51   D4     0170          CLRL    R1                       ; Restart with R1 set to zero
                        05     0172          RSB                              ; Get back to work
                               0173
                               0173          END_MARK_POINT         CVTLP_M_STATE
                               0173
                               0173          .END
```

```
...PC...                         = 0000009C
...RESTART_PC...                 = 0000009C
...ROPRAND...                    = 0000001C R      02
CVTLP_1                            00000101 R      02
CVTLP_1_RESTART                  = 00000001
CVTLP_2                            00000107 R      02
CVTLP_2_RESTART                  = 00000002
CVTLP_3                            0000010D R      02
CVTLP_3_RESTART                  = 00000003
CVTLP_4                            00000113 R      02
CVTLP_4_RESTART                  = 00000004
CVTLP_5                            00000119 R      02
CVTLP_5_RESTART                  = 00000005
CVTLP_6                            0000011F R      02
CVTLP_6_RESTART                  = 00000006
CVTLP_7                            00000125 R      02
CVTLP_7_RESTART                  = 00000007
CVTLP_ACCVIO                       000000DE R      02
CVTLP_B_DELTA_PC                 - 0000000B
CVTLP_B_SAVED_PSW                = 00000006
CVTLP_B_SAVED_R4                 = 00000004
CVTLP_B_SAVED_R5                 = 00000005
CVTLP_B_STATE                    = 00000007
CVTLP_M_FPD                      = 00000008
CVTLP_M_STATE                    = 00000007
CVTLP_S_STATE                    = 00000003
CVTLP_V_FPD                      = 00000003
CVTLP_V_STATE                    = 00000000
DECIMAL$BINARY_TO_PACKED_TABLE     ********    X   00
DECIMAL_ROPRAND                    000000D5 R      02
EXCEPTION_PSL                    = 0000002C
HANDLER_TABLE_BASE                 00000000 R      04
MODULE_BASE                      = 00000000 R      02
MODULE_END                       = 00000173 R      02
PACK_M_ACCVIO                    = 00000200
PACK_M_FPD                       = 00000100
PC_TABLE_BASE                      00000000 R      03
PSL$M_C                          = 00000001
PSL$M_N                          = 00000008
PSL$M_V                          = 00000002
PSL$M_Z                          = 00000004
PSL$V_DV                         = 00000007
PSL$V_Z                          = 00000002
RESTART_PC_TABLE_BASE              00000000 R      05
RESTART_TABLE_SIZE               = 00000007
TABLE_SIZE                       = 00000007
VAX$CVTLP                          00000003 RG     02
VAX$CVTLP_RESTART                  00000145 RG     02
VAX$DECIMAL_OVERFLOW               ********    X   00
VAX$EXIT_EMULATOR                  ********    X   00
VAX$REFLECT_FAULT                  ********    X   00
VAX$ROPRAND                        ********    X   00
```

```
                                        +-----------------+
                                        ! 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
_VAX$CODE                       00000173  (   371.)  02 (   2.)    PIC   USR   CON   REL   LCL   SHR   EXE   RD   NOWRT NOVEC LONG
PC_TABLE                        0000000E  (    14.)  03 (   3.)    PIC   USR   CON   REL   LCL   SHR NOEXE   RD   NOWRT NOVEC BYTE
HANDLER_TABLE                   0000000E  (    14.)  04 (   4.)    PIC   USR   CON   REL   LCL   SHR NOEXE   RD   NOWRT NOVEC BYTE
RESTART_PC_TABLE                0000000E  (    14.)  05 (   5.)    PIC   USR   CON   REL   LCL   SHR NOEXE   RD   NOWRT NOVEC BYTE

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

Phase                     Page faults    CPU Time       Elapsed Time
-----                     -----------    --------       ------------
Initialization                    11     00:00:00.05    00:00:01.82
Command processing                73     00:00:00.53    00:00:05.30
Pass 1                           123     00:00:02.91    00:00:13.17
Symbol table sort                  0     00:00:00.11    00:00:00.11
Pass 2                           113     00:00:01.28    00:00:04.49
Symbol table output                7     00:00:00.06    00:00:00.06
Psect synopsis output              2     00:00:00.03    00:00:00.03
Cross-reference output             0     00:00:00.00    00:00:00.00
Assembler run totals             329     00:00:04.97    00:00:24.99

The working set limit was 1050 pages.
14994 bytes (30 pages) of virtual memory were used to buffer the intermediate code.
There were 10 pages of symbol table space allocated to hold 121 non-local and 19 local symbols.
623 source lines were read in Pass 1, producing 20 object records in Pass 2.
19 pages of virtual memory were used to define 17 macros.

                                     +----------------------------+
                                     ! Macro library statistics !
                                     +----------------------------+

Macro library name                        Macros defined
------------------                        --------------
_$255$DUA28:[EMULAT.OBJ]VAXMACROS.MLB;1               9
_$255$DUA28:[SYSLIB]STARLET.MLB;2                     5
TOTALS (all libraries)                               14

278 GETS were required to define 14 macros.

There were no errors, warnings or information messages.

MACRO/LIS=LIS$:VAXCVTLP/OBJ=OBJ$:VAXCVTLP MSRC$:VAXCVTLP/UPDATE=(ENH$:VAXCVTLP)+LIB$:VAXMACROS/LIB
```

0144 AH-BT13A-SE
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY



VAXHANDLR
LIS

VAXCVTLP
LIS

VAXDECIML
LIS

VAXEMULAT
LIS

VAXCVTPL
LIS

VAXEDITPC
LIS