```
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
```

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

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  VAX$CVTPL - VAX-11 Instruction Emulator for CVTPL
0000     2          .IDENT  /V04-000/
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:
0000    31 ;
0000    32 ;       VAX-11 Instruction Emulator
0000    33 ;
0000    34 ; Abstract:
0000    35 ;       The routine in this module emulates the VAX-11 packed decimal
0000    36 ;       CVTPL instruction. This procedure can be a part of an emulator
0000    37 ;       package or can be called directly after the input parameters
0000    38 ;       have been loaded into the architectural registers.
0000    39 ;
0000    40 ;       The input parameters to this routine are the registers that
0000    41 ;       contain the intermediate instruction state.
0000    42 ;
0000    43 ; Environment:
0000    44 ;
0000    45 ;       This routine runs at any access mode, at any IPL, and is AST
0000    46 ;       reentrant.
0000    47 ;
0000    48 ; Author:
0000    49 ;
0000    50 ;       Lawrence J. Kenah
0000    51 ;
0000    52 ; Creation Date
0000    53 ;
0000    54 ;       18 October 1983
0000    55 ;
0000    56 ; Modified by:
0000    57 ;
```

```
0000     58 ;       V01-009 LJK0034         Lawrence J. Kenah       16-Jul-1984
0000     59 ;               Fix several bugs in restart logic.
0000     60 ;
0000     61 ;               Use R4 instead of R7 as dispatch register for restart routine.
0000     62 ;               There is a single code path where R7 contains useful data.
0000     63 ;               Insure that the contents of R7 are preserved across the
0000     64 ;                 occurrence of the CVTPL_5 access violation.
0000     65 ;               Use special restart path for CVTPL_6.
0000     66 ;               Fix recalculation of srcaddr.
0000     67 ;               Use saved R2 when saving condition codes.
0000     68 ;
0000     69 ;       V01-008 LJK0033         Lawrence J. Kenah       6-Jul-1984
0000     70 ;               Add R10 to register mask used along error path when the
0000     71 ;               digit count is larger than 31.
0000     72 ;
0000     73 ;       V01-007 LJK0032         Lawrence J. Kenah       5-Jul-1984
0000     74 ;               Fix restart routine to take into account the fact that restart
0000     75 ;               codes are based at one when computing restart PC.
0000     76 ;
0000     77 ;       V01-006 LJK0030         Lawrence J. Kenah       20-Jun-1984
0000     78 ;               Load access violation handler address into R10 before
0000     79 ;               any useful work (like memory accesses) gets done.
0000     80 ;
0000     81 ;       V01-005 LJK0029         Lawrence J. Kenah       24-May-1984
0000     82 ;               Fix stack offset calculations in exit code when V-bit is set
0000     83 ;               to reflect the fact that seven registers (not six) have
0000     84 ;               been saved on the stack.
0000     85 ;
0000     86 ;       V01-004 LJK0024         Lawrence J. Kenah       22-Feb-1984
0000     87 ;               Add code to handle access violations. Perform minor cleanup.
0000     88 ;
0000     89 ;       V01-003 LJK0023         Lawrence J. Kenah       10-Feb-1984
0000     90 ;               Make a write to PC generate a reserved addressing mode fault.
0000     91 ;               Temporarily do the same thing for a SP destination operand
0000     92 ;               until a better solution can be figured out.
0000     93 ;
0000     94 ;       V01-002 LJK0016         Lawrence J. Kenah       28-Nov-1983
0000     95 ;               Algorithm was revised to work with digit pairs. Overflow check
0000     96 ;               was modified to account for -2,147,483,648.
0000     97 ;
0000     98 ;       V01-001 LJK0013         Lawrence J. Kenah       17-Nov-1983
0000     99 ;               The emulation code for CVTPL was moved into a separate module.
0000    100 ;--
```

```
                                                B 4
0000    102          .SUBTITLE       Declarations
0000    103
0000    104 ; Include files:
0000    105
0000    106          .NOCROSS                        ; No cross reference for these
0000    107          .ENABLE         SUPPRESSION     ; No symbol table entries either
0000    108
0000    109          CVTPL_DEF                       ; Bit fields in CVTPL registers
0000    11           PACK_DEF                        ; Stack usage by exception handler
0000    11           STACK_DEF                       ; Stack usage for original exception
0000    112
0000    113          $PSLDEF                         ; Define bit fields in PSL
0000    114          $SRMDEF                         ; Define arithmetic trap codes
0000    115
0000    116          .DISABLE        SUPPRESSION     ; Turn on symbol table again
0000    117          .CROSS                          ; Cross reference is OK now
0000    118
0000    119 ; External declarations
0000    120
0000    121          .DISABLE        GLOBAL
0000    122
0000    123          .EXTERNAL -
0000    124                          DECIMAL$BOUNDS_CHECK,-
0000    125                          DECIMAL$PACKED_TO_BINARY_TABLE,-
0000    126                          DECIMAL$STRIP_ZEROS_RO_RT
0000    127
0000    128          .EXTERNAL -
0000    129                          VAX$EXIT_EMULATOR,-
0000    130                          VAX$REFLECT_FAULT,-
0000    131                          VAX$REFLECT_TRAP,-
0000    132                          VAX$RADRMOD,-
0000    133                          VAX$ROPRAND
0000    134
0000    135 ; PSECT Declarations:
0000    136
0000    137          .DEFAULT        DISPLACEMENT , WORD
0000    138
00000000    139      .PSECT _VAX$CODE PIC, USR, CON, REL, LCL, SHR, EXE, RD, NOWRT, LONG
0000    140
0000    141          BEGIN_MARK_POINT        RESTART
```

```
0000    143          .SUBTITLE       VAX$CVTPL - Convert Packed to Long
0000    144 ;+
0000    145 ; Functional Description:
0000    146 ;
0000    147 ;       The source string specified by the  source  length  and  source  address
0000    148 ;       operands  is  converted  to  a  longword  and the destination operand is
0000    149 ;       replaced by the result.
0000    150 ;
0000    151 ; Input Parameters:
0000    152 ;
0000    153 ;       R0 - srclen.rw          Length of input decimal string
0000    154 ;       R1 - srcaddr.ab         Address of input packed decimal string
0000    155 ;       R3 - dst.wl             Address of longword to receive converted string
0000    156 ;
0000    157 ;       Note that the CVTPL instruction is the only instruction in the
0000    158 ;       emulator package that has an cperand type of .wx. This operand type
0000    159 ;       needs special treatment if the operand is to be written into a general
0000    160 ;       register. The following convention is established. If the destination
0000    161 ;       is anything other than a general register (addressing mode 5), then R3
0000    162 ;       contains the address of the destination. If the destination is a
0000    163 ;       general register, then R3 contains the ones complement of the register
0000    164 ;       number. Note that if this is interpreted as an address, then R3 points
0000    165 ;       to the last page of so-called S1 space, the reserved half of system
0000    166 ;       virtual address space. The algorithmic specification of this
0000    167 ;       convention is as follows.
0000    168 ;
0000    169 ;               IF R3 <31:04> NEQU ^XFFFFFFF
0000    170 ;                 THEN
0000    171 ;                   R3 contains the address of the destination operand
0000    172 ;                 ELSE
0000    173 ;                   R3 contains the ones complement of the register number
0000    174 ;                   of the single register to be loaded with the result
0000    175 ;                   of the conversion
0000    176 ;
0000    177 ;               That is,
0000    178 ;
0000    179 ;                       R3 = FFFFFFFF  ==>   R0 <- result
0000    180 ;                       R3 = FFFFFFFE  ==>   R1 <- result
0000    181 ;                              .
0000    182 ;                              .
0000    183 ;                       R3 = FFFFFFF4  ==>  R11 <- result
0000    184 ;                              .
0000    185 ;                              .
0000    186 ;
0000    187 ;               Note that any "S1 address" in R3 on input other than
0000    188 ;               FFFFFFFF through FFFFFFF0 will cause a length access
0000    189 ;               violation.
0000    190 ;
0000    191 ; Output Parameters:
0000    192 ;
0000    193 ;       R0 = 0
0000    194 ;       R1 = Address of byte containing most significant digit of
0000    195 ;            the source string
0000    196 ;       R2 = 0
0000    197 ;       R3 = 0
0000    198 ;
0000    199 ; Condition Codes:
```

```
                                0000   200 ;
                                0000   201 ;       N <- output longword LSS 0
                                0000   202 ;       Z <- output longword EQL 0
                                0000   203 ;       V <- integer overflow
                                0000   204 ;       C <- 0
                                0000   205 ;
                                0000   206 ; Register Usage:
                                0000   207 ;
                                0000   208 ;       This routine uses R0 through R7. The condition codes are recorded
                                0000   209 ;       in R2 as the routine executes. In addition, R10 serves its usual
                                0000   210 ;       purpose by pointing to the access violation handler.
                                0000   211 ;-
                                0000   212
                                0000   213         .ENABLE LOCAL_BLOCK
                                0000   214
                                0000   215         ASSUME CVTPL_B_STATE EQ 2       ; Make sure we test the right FPD bit
                                0000   216
                     01B0   31  0000   217 2$:     BRW     VAX$CVTPL_RESTART       , Restart somewhere else
                                0003   218
                                0003   219 VAX$CVTPL::
                  F9 50   17   E0  0003   220         BBS     #<CVTPL_V_FPD+16>,R0,2$ ; Branch if this is a restart
                     04F2 8F   BB  0007   221         PUSHR   #^M<R1,R4,R5,R6,R7,R10> ; Save some registers
                                000B   222         ESTABLISH_HANDLER       -
                                000B   223                         CVTPL_ACCVIO    ; Load R10 with handler address
                        52   DC  0010   224         MOVPSL  R2                      ; Get current PSL
                     52   0F   8A  0012   225         BICB2   #<PSL$M_N!PSL$M_Z!PSL$M_V!PSL$M_C>,R2    ; Clear condition codes
                        56   D4  0015   226         CLRL    R6                      ; Assume result is zero
                                0017   227         ROPRAND_CHECK   R0              ; Insure that R0 LEQU 31
                        76   13  0022   228         BEQL    60$                     ; All done if string has zero length
                                0024   229         MARK_POINT      CVTPL_1 , RESTART
                           FFD9' 30  0024   230         BSBW    DECIMAL$STRIP_ZEROS_R0_R1       ; Eliminate leading zeros from input
                  50   50   FF 8F   78  0027   231         ASHL    #-1,R0,R0               ; Convert digit count to byte count
                           2B   13  002C   232         BEQL    30$                     ; Skip loop if single digit
                                002E   233
                                002E   234 ; The first digit pair sets up the initial value of the result.
                                002E   235
                                002E   236         MARK_POINT      CVTPL_2 , RESTART
                        55   81   9A  002E   237         MOVZBL  (R1)+,R5                ; Get first digit pair
                     56   0000'CF45   9A  0031   238         MOVZBL  DECIMAL$PACKED_TO_BINARY_TABLE[R5],R6
                                0037   239                                         ; Convert to binary number
                                0037   240
                                0037   241 ; The SOBGTR instruction at the bottom of the loop can be used to decrement
                                0037   242 ; the byte count and test whether this is the special case of an initial
                                0037   243 ; digit count of two or three. Note that this loop does not attempt to
                                0037   244 ; optimize the case where the V-bit is already set.
                                0037   245
                           1D   11  0037   246         BRB     20$                     ; Join the loop at the bottom
                                0039   247
                                0039   248         MARK_POINT      CVTPL_3 , RESTART
                        55   81   9A  0039   249 10$:    MOVZBL  (R1)+,R5                ; Get next digit pair
                     55   0000'CF45   9A  003C   250         MOVZBL  DECIMAL$PACKED_TO_BINARY_TABLE[R5],R5
                                0042   251                                         ; Convert to binary number
56   55   56   00000064 8F   7A  0042   252         EMUL    #100,R6,R5,R6           ; Blend this latest with previous result
                                004B   253
                                004B   254 ; Check all of R7 and R6<31> for nonzero. Unconditionally clear R6<31>.
                                004B   255
                     04 56   1F   E4  004B   256         BBSC    #31,R6,15$              ; Branch if overflow into R6<31>
```

```
                                57   D5  004F   257          TSTL    R7                      ; Anything into upper longword
                                03   13  0051   258          BEQL    20$                     ; Branch if OK
                             52 02   88  0053   259 15$:     BISB    #PSL$M_V,R2             ; Set saved V-bit
                               E0 50 F5  0056   260 20$:     SOBGTR  R0,10$                  ; Continue for rest of whole digit pairs
                                     0059   261
                                     0059   262 ; The final (least significant) digit is handled in a slightly different
                                     0059   263 ; fashion. This has an advantage in that the final overflow check is different
                                     0059   264 ; from the check that is made inside the loop. That check can be made quickly
                                     0059   265 ; without concern for the final digit special cases.
                                     0059   266
                                     0059   267          MARK_POINT      CVTPL_4 , RESTART
                55   61   04   04 EF 0059   268 30$:     EXTZV   #4,#4,(R1),R5           ; Get least significant digit
                5o   55   56   0A 7A 005E   269          EMUL    #10,R6,R5,R6            ; Blend in with previous result
                                     0063   270
                                     0063   271 ; This overflow check differs from the one inside the loop in three ways.
                                     0063   272 ;
                                     0063   273 ;       The check for nonzero R7 precedes the test of R6<31>.
                                     0063   274 ;
                                     0063   275 ;     o The high order bit of R6 is left alone. (If overflow occurs, the
                                     0063   276 ;       complete 32-bit contents of R6 need to be preserved.)
                                     0063   277 ;
                                     0063   278 ;     o A special check is made to see if the 64-bit result is identically
                                     0063   279 ;       equal to
                                     0063   280 ;
                                     0063   281 ;               R6 = 80000000
                                     0063   282 ;               R7 = 00000000
                                     0063   283 ;
                                     0063   284 ;     o If this is true AND the input sign is minus, then the overflow bit
                                     0063   285 ;       needs to be turned off. This unusual result is passed to the following
                                     0063   286 ;       code by means of a zero in R7. All other results cause nonzero R7
                                     0063   287 ;       (including the case where the V-bit was already set).
                                     0063   288 ;
                                     0063   289 ; Note that the check for V-bit previously set is the single additional
                                     0063   290 ; instruction that must execute in the normal (V-bit clear) case to test
                                     0063   291 ; for the extraordinarily rare case of -2,147,483,648.
                                     0063   292
                                57   D5  0063   293          TSTL    R7                      ; Overflow into second longword?
                                11   12  0065   294          BNEQ    36$                     ; Branch if overflow
                          0B 52 01   E0  0067   295          BBS     #PSL$V_V,R2,33$         ; Set R7 to nonzero if V-bit already set
                80000000 8F 56   D1  006B   296          CMPL    R6,#^X80000000          ; Peculiar check for R6<31> NEQ zero
                                07   1F  0072   297          BLSSU   40$                     ; Branch if no overflow at all
                                02   13  0074   298          BEQL    36$                     ; Leave R7 alone in special case
                                57   D6  0076   299 33$:     INCL    R7                      ; Set R7 to nonzero in all other cases
                             52 02   88  0078   300 36$:     BISB    #PSL$M_V,R2             ; Set saved V-bit
                                     007B   301
                                     007B   302 ; All of the input digits have been processed, Get the sign of the input
                                     007B   303 ; string and complete the instruction processing.
                                     007B   304
                                     007B   305          MARK_POINT      CVTPL_5 , RESTART
                55   61   F0 8F 8B 007B   306 40$:     BICB3   #^B11110000,(R1),R5     ; Get sign "digit"
                                     0080   307
                                     0080   308          CASE    R5,LIMIT=#10,TYPE=B,<-  ; Dispatch on sign
                                     0080   309                  60$,-                   ; 10 => +
                                     0080   310                  50$,-                   ; 11 => -
                                     0080   311                  60$,-                   ; 12 => +
                                     0080   312                  50$,-                   ; 13 => -
                                     0080   313                  60$,-                   ; 14 => +
```

```
                     0080   314                60$,-                   ; 15 => +
                     0080   315                >
                     0090   316
                     0090   317 ; Note that negative zero is not a problem in this instruction because the
                     0090   318 ; longword result will simply be zero, independent of the input sign.
                     0090   319
56   56   CE         0090   320 50$:    MNEGL   R6,R6                   ; Change sign on negative input
     57   D5         0093   321        TSTL    R7                      ; Was input -2,147,483,648?
     03   12         0095   322        BNEQ    60$                     ; Nope, leave V-bit alone
52   02   8A         0097   323        BICB    #PSL$M_V,R2             ; Clear saved V-bit
51   8E   D0         009A   324 60$:    MOVL    (SP)+,R1                ; Restore original value of R1
     7E   7C         009D   325        CLRQ    -(SP)                   ; Set saved R2 and R3 to zero
                     009F   326
                     009F   327 ; If R3 contains the ones complement of a number between 0 and 15, then the
                     009F   328 ; destination is a general register. Special processing is required to
                     009F   329 ; correctly restore registers, store the result in a register, and set the
                     009F   330 ; condition codes.
                     009F   331
57   53   D2         009F   332        MCOML   R3,R7                   ; Set up R7 for limit check with CASE
                     00A2   333        CASE    R7,LIMIT=#0,TYPE=L,<-   ; See if R7 contains a register number
                     00A2   334
                     00A2   335                100$,-                  ; R0  --  Store into R0 via POPR
                     00A2   336                100$,-                  ; R1  --  Store into R1 via POPR
                     00A2   337
                     00A2   338                110$,-                  ; R2  --  Store in saved R2 on stack
                     00A2   339                110$,-                  ; R3  --  Store in saved R3 on stack
                     00A2   340                110$,-                  ; R4  --  Store in saved R4 on stack
                     00A2   341                110$,-                  ; R5  --  Store in saved R5 on stack
                     00A2   342                110$,-                  ; R6  --  Store in saved R6 on stack
                     00A2   343                110$,-                  ; R7  --  Store in saved R7 on stack
                     00A2   344
                     00A2   345                100$,-                  ; R8  --  Store into R8 via POPR
                     00A2   346                100$,-                  ; R9  --  Store into R9 via POPR
                     00A2   347                120$,-                  ; R10 --  Store in saved R10 on stack
                     00A2   348                100$,-                  ; R11 --  Store into R11 via POPR
                     00A2   349                100$,-                  ; AP  --  Store into AP via POPR
                     00A2   350                100$,-                  ; FP  --  Store into FP via POPR
                     00A2   351
                     00A2   352 ; The result of specifying PC as a destination operand is defined to be
                     00A2   353 ; UNPREDICTABLE in the VAX architecture. In addition, it is difficult (but
                     00A2   354 ; not impossible) for this emulator to modify SP because it is using the
                     00A2   355 ; stack for local storage. We will generate a reserved addressing mode fault
                     00A2   356 ; if PC is specified as the destination operand. We will also temporarily
                     00A2   357 ; generate a reserved addressing mode fault if SP is specified as the
                     00A2   358 ; destination operand.
                     00A2   359
                     00A2   360                CVTPL_RADRMOD,-         ; SP  --  Reserved addressing mode
                     00A2   361                CVTPL_RADRMOD,-         ; PC  --  Reserved addressing mode
                     00A2   362                >
                     00C6   363
                     00C6   364 ; If we drop through the CASE instruction, then R3 contains the address of
                     00C6   365 ; the destination operand. This includes system space addresses in the range
                     00C6   366 ; C0000000 to FFFFFFFF other than the ones complements of 0 through 15
                     00C6   367 ; (FFFFFFF0 to FFFFFFFF). The next instruction will cause an access violation
                     00C6   368 ; for all such illegal system space addresses.
                     00C6   369
                     00C6   370          MARK_POINT      CVTPL_6 , RESTART
```

```
                63   56   D0   00C6   371          MOVL    R6,(R3)                 ; Store result and set condition codes
                                00C9   372
                                00C9   373 ; This is the exit path for this routine. The result has already been stored.
                                00C9   374 ; The condition codes are set and the saved registers restored. The BICPSW
                                00C9   375 ; instruction is necessary because the various instructions that stored the
                                00C9   376 ; result (MOVL, PUSHL, etc.) do not affect the C-bit and the C-bit must be
                                00C9   377 ; clear on exit from this routine.
                                00C9   378
                      01   B9   00C9   379 70$:    BICPSW  #PSL$M_C                ; Insure that C-bit is clear on exit
                      52   B8   00CB   380          BISPSW  R2                      ; Set saved V-bit
                  0E  52   01   E0   00CD   381          BBS     #PSL$V_V,R2,75$         ; Step out of line for overflow check
                    04FC   8F   BA   00D1   382          POPR    #^M<R2,R3,R4,R5,R6,R7,R10>      ; Restore saved registers
                                00D5   383                                          ;  and clear R2 and R3
                                05   00D5   384          RSB
                                00D6   385
                      0F   B9   00D6   386 72$:    BICPSW  #<PSL$M_N!PSL$M_Z!PSL$M_V!PSL$M_C>      ; Clear condition codes
                      53   B8   00D8   387          BISPSW  R3                      ; Set relevant condition codes
                    04FC   8F   BA   00DA   388          POPR    #^M<R2,R3,R4,R5,R6,R7,R10>      ; Restore saved registers
                                05   00DE   389          RSB
                                00DF   390
                                00DF   391 ; If the V-bit is set and decimal traps are enabled (IV-bit is set), then
                                00DF   392 ; a decimal overflow trap is generated. Note that the IV-bit can be set in
                                00DF   393 ; the current PSL or, if this routine was entered as the result of an emulated
                                00DF   394 ; instruction exception, in the saved PSL on the stack.
                                00DF   395
                      53   DC   00DF   396 75$:    MOVPSL  R3                      ; Save current condition codes
                   14 52   05   E0   00E1   397          BBS     #PSL$V_IV,R2,78$        ; Report exception if current IV-bit set
                52    0000'CF   9E   00E5   398          MOVAB   VAX$EXIT_EMULATOR,R2    ; Set up R2 for PIC address comparison
                  1C AE   52   D1   00EA   399          CMPL    R2,<4*7>(SP)            ; Is return PC EQLU VAX$EXIT_EMULATOR ?
                         E6   12   00EE   400          BNEQU   72$                     ; No. Simply return V-bit set
                E1 4C AE   05   E1   00F0   401          BBC     #PSL$V_IV,<<4*<7+1>>+EXCEPTION_PSL>(SP),72$
                                00F5   402                                          ; Only return V-bit if IV-bit is clear
                                00F5   403
                      0F   B9   00F5   404          BICPSW  #<PSL$M_N!PSL$M_Z!PSL$M_V!PSL$M_C>      ; Clear condition codes
                      53   B8   00F7   405          BISPSW  R3                      ; Set relevant condition codes
                                00F9   406
            04FC  8F   BA   00F9   407 78$:    POPR    #^M<R2,R3,R4,R5,R6,R7,R10>      ; Otherwise, restore registers
                                00FD   408
                                00FD   409 ; ... drop into INTEGER_OVERFLOW
```

```
                     00FD   411 ;+
                     00FD   412 ; This code path is entered if the result is too large to fit into a longword
                     00FD   413 ; and integer overflow exceptions are enabled. The final state of the
                     00FD   414 ; instruction, including the condition codes, is entirely in place.
                     00FD   415 ;
                     00FD   416 ; Input Parameter:
                     00FD   417 ;
                     00FD   418 ;       (SP) - Return PC
                     00FD   419 ;
                     00FD   420 ; Output Parameters:
                     00FD   421 ;
                     00FD   422 ;       0(SP) - SRM$K_INT_OVF_T (Arithmetic trap code)
                     00FD   423 ;       4(SP) - Final state PSL
                     00FD   424 ;       8(SP) - Return PC
                     00FD   425 ;
                     00FD   426 ; Implicit Output:
                     00FD   427 ;
                     00FD   428 ;       Control passes through this code to VAX$REFLECT_TRAP.
                     00FD   429 ;-
                     00FD   430
                     00FD   431 INTEGER_OVERFLOW:
          7E   DC    00FD   432         MOVPSL  -(SP)                   ; Save final PSL on stack
          01   DD    00FF   433         PUSHL   #SRM$K_INT_OVF_T        ; Store arithmetic trap code
      FEFC'    31    0101   434         BRW     VAX$REFLECT_TRAP        ; Report exception
                     0104   435
                     0104   436 ;+
                     0104   437 ; The destination address is a general register. R3 contains the ones
                     0104   438 ; complement of the register number of the general register that is to be
                     0104   439 ; loaded with the result. Note that the result must be stored in such a way
                     0104   440 ; that restoring the saved registers does not overwrite the destination.
                     0104   441 ;
                     0104   442 ; The algorithm that accomplishes a correct store of the result with the
                     0104   443 ; accompanying setting of the condition codes is as follows.
                     0104   444 ;
                     0104   445 ;       If the register is in the range R2 through R7 or R10
                     0104   446 ;         THEN
                     0104   447 ;           store the result on the stack over that saved register
                     0104   448 ;           (note that this store sets condition codes, except the C-bit)
                     0104   449 ;         ELSE
                     0104   450 ;           construct a register save mask from the register number
                     0104   451 ;           store result on the top of the stack
                     0104   452 ;           (note that this store sets condition codes, except the C-bit)
                     0104   453 ;           POPR the result using the mask in R3
                     0104   454 ;       ENDIF
                     0104   455 ;       restore saved registers
                     0104   456 ;-
                     0104   457
                     0104   458 ; R7 contains 0, 1, 8, 9, 11, 12, or 13. We will use the bit number to
                     0104   459 ; create a register save mask for the appropriate register. Note that #1
                     0104   460 ; is the source operand and R7 is the shift count in the next instruction.
                     0104   461
    53   01   57    78  0104   462 100$:   ASHL    R7,#1,R3                ; R3 contains mask for single register
                56   DD  0108   463         PUSHL   R6                      ; Store result and set condition codes
                53   BA  010A   464         POPR    R3                      ; Restore result into correct register
                BB   11  010C   465         BRB     70$                     ; Restore registers and return
                         010E   466
                         010E   467 ; R7 contains 2, 3, 4, 5, 6, or 7
```

```
                     010E   468
F8 AE47   56   D0   010E   469 110$:   MOVL    R6,-8(SP)[R7]           ; Store result over saved register
          B4   11   0113   470         BRB     70$                     ; Restore registers and return
                    0115   471
                    0115   472 ; R7 contains a 10
                    0115   473
 18 AE    56   D0   0115   474 120$:   MOVL    R6,24(SP)               ; Store result over saved register
          AE   11   0119   475         BRB     70$                     ; Restore registers and return
                    011B   476
                    011B   477         .DISABLE        LOCAL_BLOCK
```

```
                     011B   479          .SUBTITLE       DECIMAL_ROPRAND
                     011B   480 ;-
                     011B   481 ; Functional Description:
                     011B   482 ;
                     011B   483 ;       This routine receives control when a digit count larger than 31
                     011B   484 ;       is detected. The exception is architecturally defined as an
                     011B   485 ;       abort so there is no need to store intermediate state. The digit
                     011B   486 ;       count is made after registers are saved. These registers must be
                     011B   487 ;       restored before reporting the exception.
                     011B   488 ;
                     011B   489 ; Input Parameters:
                     011B   490 ;
                     011B   491 ;       00(SP) - Saved R1
                     011B   492 ;       04(SP) - Saved R4
                     011B   493 ;       08(SP) - Saved R5
                     011B   494 ;       12(SP) - Saved R6
                     011B   495 ;       16(SP) - Saved R7
                     011B   496 ;       20(SP) - Saved R10
                     011B   497 ;       24(SP) - Return PC from VAX$CVTPL routine
                     011B   498 ;
                     011B   499 ; Output Parameters:
                     011B   500 ;
                     011B   501 ;       00(SP) - Offset in packed register array to delta PC byte
                     011B   502 ;       04(SP) - Return PC from VAX$CVTPL routine
                     011B   503 ;
                     011B   504 ; Implicit Output:
                     011B   505 ;
                     011B   506 ;       This routine passes control to VAX$ROPRAND where further
                     011B   507 ;       exception processing takes place.
                     011B   508 ;-
                     011B   509
                     011B   510 DECIMAL_ROPRAND:
04F2 8F    BA        011B   511          POPR    #^M<R1,R4,R5,R6,R7,R10> ; Restore saved registers
     03    DD        011F   512          PUSHL   #CVTPL_B_DELTA_PC       ; Store offset to delta PC byte
  FEDC'    31        0121   513          BRW     VAX$ROPRAND             ; Pass control along
```

```
                     0124    515          .SUBTITLE       CVTPL_RADRMOD
                     0124    516 ;-
                     0124    517 ; Functional Description:
                     0124    518 ;
                     0124    519 ;       This routine receives control when PC or SP is used as the destination
                     0124    520 ;       of a CVTPL instruction. The reaction to this is not architecturally
                     0124    521 ;       defined so we are somewhat free in how we handle it. We currently
                     0124    522 ;       generate a RADRMOD abort with R0 containing the correct 32-bit result.
                     0124    523 ;       In the future, we may make this instruction restartable following this
                     0124    524 ;       exception.
                     0124    525 ;
                     0124    526 ; Input Parameters:
                     0124    527 ;
                     0124    528 ;       R0 - Zero
                     0124    529 ;       R1 - Address of source decimal string
                     0124    530 ;       R2 - Contains overflow indication in R2<PSL$V_V>
                     0124    531 ;       R3 - Register number in ones complement form
                     0124    532 ;               R3 = -15 => PC was destination operand
                     0124    533 ;               R3 = -14 => SP was destination operand
                     0124    534 ;       R4 - scratch
                     0124    535 ;       R5 - scratch
                     0124    536 ;       R6 - Correct 32-bit result
                     0124    537 ;       R7 - scratch
                     0124    538 ;
                     0124    539 ;       00(SP) - Saved R2 (contains zero)
                     0124    540 ;       04(SP) - Saved R3 (contains zero)
                     0124    541 ;       08(SP) - Saved R4
                     0124    542 ;       12(SP) - Saved R5
                     0124    543 ;       16(SP) - Saved R6
                     0124    544 ;       20(SP) - Saved R7
                     0124    545 ;       24(SP) - SAVED R10
                     0124    546 ;       28(SP) - Return PC from VAX$xxxxxx routine
                     0124    547 ;
                     0124    548 ; Output Parameters:
                     0124    549 ;
                     0124    550 ;       R0 - Correct 32-bit result
                     0124    551 ;
                     0124    552 ;       R1, R2, and R3 are unchanged from their input values.
                     0124    553 ;
                     0124    554 ;       R4 through R7 and R10 are restored from the stack.
                     0124    555 ;
                     0124    556 ;       00(SP) - Offset in packed register array to delta PC byte
                     0124    557 ;       04(SP) - Return PC from VAX$xxxxxx routine
                     0124    558 ;
                     0124    559 ; Implicit Output:
                     0124    560 ;
                     0124    561 ;       This routine passes control to VAX$RADRMOD where further
                     0124    562 ;       exception processing takes place.
                     0124    563 ;-
                     0124    564
                     0124    565 CVTPL_RADRMOD:
5E   08   C0         0124    566          ADDL    #8,SP                   ; Discard "saved" R2 and R3
50   56   D0         0127    567          MOVL    R6,R0                   ; Remember final result
04F0 8F   BA         012A    568          POPR    #^M<R4,R5,R6,R7,R10>    ; Restore saved registers
     03   DD         012E    569          PUSHL   #CVTPL_B_DELTA_PC       ; Store offset to delta PC byte
   FECD'  31         0130    570          BRW     VAX$RADRMOD             ; Pass control along
```

```
                     0133   572          .SUBTITLE       CVTPL_ACCVIO - Reflect an Access Violation
                     0133   573 ;+
                     0133   574 ; Functional Description:
                     0133   575 ;
                     0133   576 ;       This routine receives control when an access violation occurs while
                     0133   577 ;       executing within the VAX$CVTPL emulator routine.
                     0133   578 ;
                     0133   579 ;       The routine header for ASHP_ACCVIO in module VAX$ASHP contains a
                     0133   580 ;       detailed description of access violation handling for the decimal
                     0133   581 ;       string instructions. This routine differs from most decimal
                     0133   582 ;       instruction emulation routines in that it preserves intermediate
                     0133   583 ;       results if an access violation occurs. This is accomplished by
                     0133   584 ;       storing the number of the exception point, as well as intermediate
                     0133   585 ;       arithmetic results, in the registers R0 through R3.
                     0133   586 ;
                     0133   587 ; Input Parameters:
                     0133   588 ;
                     0133   589 ;       See routine ASHP_ACCVIO in module VAX$ASHP
                     0133   590 ;
                     0133   591 ; Output Parameters:
                     0133   592 ;
                     0133   593 ;       See routine ASHP_ACCVIO in module VAX$ASHP
                     0133   594 ;-
                     0133   595
                     0133   596 CVTPL_ACCVIO:
          52   D4   0133   597          CLRL    R2                      ; Initialize the counter
    FEC7 CF   9F   0135   598          PUSHAB  MODULE_BASE             ; Store base address of this module
01F6'CF   9F   0139   599          PUSHAB  MODULE_END              ; Store module end address
   FEC0'  30   013D   600          BSBW    DECIMAL$BOUNDS_CHECK    ; Check if PC is inside the module
5E   04   C0   0140   601          ADDL    #4,SP                   ; Discard end address
51   8E   C2   0143   602          SUBL2   (SP)+,R1                ; Get PC relative to this base
                     0146   603
0000'CF42   51   B1   0146   604 10$:    CMPW    R1,PC_TABLE_BASE[R2]    ; Is this the right PC?
          07   13   014C   605          BEQL    30$                     ; Exit loop if true
    F4 52   06   F2   014E   606          AOBLSS  #TABLE_SIZE,R2,10$      ; Do the entire table
                     0152   607
                     0152   608 ; If we drop through the dispatching based on PC, then the exception is not
                     0152   609 ; one that we want to back up. We simply reflect the exception to the user.
                     0152   610
          0F   BA   0152   611 20$:    POPR    #^M<R0,R1,R2,R3>        ; Restore saved registers
          05   0154   612          RSB                             ; Return to exception dispatcher
                     0155   613
                     0155   614 ; The exception PC matched one of the entries in our PC table. R2 contains
                     0155   615 ; the index into both the PC table and the handler table. R1 has served
                     0155   616 ; its purpose and can be used as a scratch register.
                     0155   617
51   0000'CF42   3C   0155   618 30$:    MOVZWL  HANDLER_TABLE_BASE[R2],R1       ; Get the offset to the handler
    FEA0 CF41   17   015B   619          JMP     MODULE_BASE[R1]         ; Pass control to the handler
                     0160   620
                     0160   621 ; In all of the instruction-specific routines, the state of the stack
                     0160   622 ; will be shown as it was when the exception occurred. All offsets will
                     0160   623 ; be pictured relative to R0.
```

```
0160   625          .SUBTITLE       Context-Specific Access Violation Handling for VAX$CVTPL
0160   626 ;+
0160   627 ; Functional Description:
0160   628 ;
0160   629 ;       The intermediate state of the instruction is packed into registers R0
0160   630 ;       through R3 and control is passed to VAX$REFLECT_FAULT that will, in
0160   631 ;       turn, reflect the access violation back to the user. The intermediate
0160   632 ;       state reflects the point at which the routine was executing when the
0160   633 ;       access violation occurred.
0160   634 ;
0160   635 ; Input Parameters:
0160   636 ;
0160   637 ;       R0 - Address of top of stack when access violation occurred
0160   638 ;
0160   639 ;       00(SP) - Saved R0 (restored by VAX$HANDLER)
0160   640 ;       04(SP) - Saved R1
0160   641 ;       08(SP) - Saved R2
0160   642 ;       12(SP) - Saved R3
0160   643 ;
0160   644 ;       See individual entry points for details
0160   645 ;
0160   646 ; Output Parameters:
0160   647 ;
0160   648 ;       R0 - Address of return PC from VAX$CVTPL
0160   649 ;       R1 - Byte offset to delta-PC in saved register array
0160   650 ;               (PACK_V_FPD and PACK_M_ACCVIO set to identify exception)
0160   651 ;
0160   652 ;       See list of input parameters for CVTPL_RESTART for a description of the
0160   653 ;       contents of the packed register array.
0160   654 ;
0160   655 ; Implicit Output:
0160   656 ;
0160   657 ;       R4, R5, R6, R7, and R10 are restored to the values that they had
0160   658 ;       when VAX$CVTPL was entered.
0160   659 ;-
0160   660
0160   661          .ENABLE         LOCAL_BLOCK
0160   662
0160   663 ;+
0160   664 ; CVTPL_1
0160   665 ;
0160   666 ; An access violation occurred in subroutine STRIP_ZEROS while scanning the
0160   667 ; source string for leading zeros.
0160   668 ;
0160   669 ;       R0  - Updated digit or byte count in source string
0160   670 ;       R1  - Address of current byte in source string
0160   671 ;       R2  - Condition codes reflecting result
0160   672 ;       R3  - Address of destination (unchanged from input value)
0160   673 ;       R6  - Intermediate (or final) longword result
0160   674 ;
0160   675 ;       00(R0) - Return PC from STRIP_ZEROS
0160   676 ;       04(R0) - Original value of R1 (scraddr)
0160   677 ;       08(R0) - Saved R4
0160   678 ;       12(R0) - Saved R5
0160   679 ;       16(R0) - Saved R6
0160   680 ;       20(R0) - Saved R7
0160   681 ;       24(R0) - Saved R10
```

```
                     0160   682 ;       28(R0) - Return PC from VAX$CVTPL routine
                     0160   683 ;-
                     0160   684
                     0160   685 CVTPL_1:
50   04   C0         0160   686         ADDL    #4,R0                   ; Discard return PC from STRIP_ZEROS
54   01   90         0163   687         MOVB    #CVTPL_1_RESTART,R4     ; Store code that locates exception PC
     1E   11         0166   688         BRB     10$                     ; Join common code
                     0168   689
                     0168   690 ;+
                     0168   691 ; CVTPL_2 through CVTPL_5
                     0168   692 ;
                     0168   693 ;       R0  - Updated digit or byte count in source string
                     0168   694 ;       R1  - Address of current byte in source string
                     0168   695 ;       R2  - Condition codes reflecting result
                     0168   696 ;       R3  - Address of destination (unchanged from input value)
                     0168   697 ;       R6  - Intermediate (or final) longword result
                     0168   698 ;
                     0168   699 ;       00(R0) - Original value of R1 (scraddr)
                     0168   700 ;       04(R0) - Saved R4
                     0168   701 ;       08(R0) - Saved R5
                     0168   702 ;       12(R0) - Saved R6
                     0168   703 ;       16(R0) - Saved R7
                     0168   704 ;       20(R0) - Saved R10
                     0168   705 ;       24(R0) - Return PC from VAX$CVTPL routine
                     0168   706 ;-
                     0168   707
                     0168   708 CVTPL_2:
54   02   90         0168   709         MOVB    #CVTPL_2_RESTART,R4     ; Store code that locates exception PC
     19   11         016B   710         BRB     10$                     ; Join common code
                     016D   711
                     016D   712 CVTPL_3:
54   03   90         016D   713         MOVB    #CVTPL_3_RESTART,R4     ; Store code that locates exception PC
     14   11         0170   714         BRB     10$                     ; Join common code
                     0172   715
                     0172   716 CVTPL_4:
54   04   90         0172   717         MOVB    #CVTPL_4_RESTART,R4     ; Store code that locates exception PC
     0F   11         0175   718         BRB     10$                     ; Join common code
                     0177   719
                     0177   720 CVTPL_5:
54   05   90         0177   721         MOVB    #CVTPL_5_RESTART,R4     ; Store code that locates exception PC
     0A   11         017A   722         BRB     10$                     ; Join common code
                     017C   723
                     017C   724 ;+
                     017C   725 ; CVTPL_6
                     017C   726 ;
                     017C   727 ;       R0  - Updated digit or byte count in source string
                     017C   728 ;       R1  - Address of most significant byte in source string (original srcaddr)
                     017C   729 ;       R2  - Condition codes reflecting result
                     017C   730 ;       R3  - Address of destination (unchanged from input value)
                     017C   731 ;       R6  - Intermediate (or final) longword result
                     017C   732 ;
                     017C   733 ;       00(R0) - Zero (will be restored to R2)
                     017C   734 ;       04(R0) - Zero (will be restored to R3)
                     017C   735 ;       08(R0) - Saved R4
                     017C   736 ;       12(R0) - Saved R5
                     017C   737 ;       16(R0) - Saved R6
                     017C   738 ;       20(R0) - Saved R7
```

```
                                017C    739 ;       24(R0) - Saved R10
                                017C    740 ;       28(R0) - Return PC from VAX$CVTPL routine
                                017C    741 ;-
                                017C    742
                                017C    743 CVTPL_6:
                50    04    C0  017C    744         ADDL    #4,R0                   ; Discard extra longword on the stack
                54    06    90  017F    745         MOVB    #CVTPL_6_RESTART,R4     ; Store code that locates exception PC
                  04  AE    D0  0182    746         MOVL    PACK_L_SAVED_R1(SP),-
                      60        0185    747                 (R0)                    ; Put "current" R1 on top of stack
                                0186    748
          51    04  AE    80  C3  0186    749 10$:    SUBL3   (R0)+,CVTPL_A_SRCADDR(SP),R1    ; Current minus initial srcaddr
                01  AE    51  90  018B    750         MOVB    R1,CVTPL_B_DELTA_SRCADDR(SP)    ; Remember it for restart
                                018F    751
                                018F    752         ASSUME CVTPL_V_SAVED_PSW EQ 0
                                018F    753
                  80  8F    89  018F    754         BISB3   #CVTPL_M_FPD,-                  ; Save current condition codes
                  08  AE        0192    755                 PACK_L_SAVED_R2(SP),-           ;  (found in saved R2)
                  02  AE        0194    756                 CVTPL_B_STATE(SP)               ;  and set internal FPD bit
                04    54    F0  0196    757         INSV    R4,#CVTPL_V_STATE,-             ; Store code that identifies
                      03        0199    758                 #CVTPL_S_STATE,-                ;  exception PC so that we
                  02  AE        019A    759                 CVTPL_B_STATE(SP)               ;  restart at correct place
             08   AE    56  D0  019C    760         MOVL    R6,CVTPL_L_RESULT(SP)           ; Save intermediate result
                                01A0    761
                                01A0    762 ; At this point, all intermediate state has been preserved in the register
                                01A0    763 ; array on the stack. We now restore the registers that were saved on entry
                                01A0    764 ; to VAX$CVTPL and pass control to VAX$REFLECT_FAULT where further exception
                                01A0    765 ; dispatching takes place.
                                01A0    766
                54    80    7D  01A0    767         MOVQ    (R0)+,R4                ; Restore R4 and R6
                56    80    7D  01A3    768         MOVQ    (R0)+,R6                ; ... and R6 and R7
                5A    80    D0  01A6    769         MOVL    (R0)+,R10               ; ... and R10
                                01A9    770
          51    00000303  8F  D0  01A9    771         MOVL    #<CVTPL_B_DELTA_PC!-    ; Indicate offset for delta PC
                                01B0    772                 PACK_M_FPD!-            ; FPD bit should be set
                                01B0    773                 PACK_M_ACCVIO>,R1       ; This is an access violation
                      FE4D'     31  01B0    774         BRW     VAX$REFLECT_FAULT       ; Continue exception handling
                                01B3    775
                                01B3    776         .DISABLE        LOCAL_BLOCK
```

```
01B3    778          .SUBTITLE       CVTPL_RESTART - Unpack and Restart CVTPL Instruction
01B3    779 ;+
01B3    780 ; Functional Description:
01B3    781 ;
01B3    782 ;       This routine receives control when a CVTPL instruction is restarted.
01B3    783 ;       The instruction state (stack and general registers) is restored to the
01B3    784 ;       state that it was in when the instruction (routine) was interrupted and
01B3    785 ;       control is passed to the PC at which the exception occurred.
01B3    786 ;
01B3    787 ; Input Parameters:
01B3    788 ;
01B3    789 ;        31               23               15               07               00
01B3    790 ;       +----------------+----------------+----------------+----------------+
01B3    791 ;       |    delta-PC    |      state     |                srclen          | : R0
01B3    792 ;       +----------------+----------------+----------------+----------------+
01B3    793 ;       |                                 srcaddr                           | : R1
01B3    794 ;       +----------------+----------------+----------------+----------------+
01B3    795 ;       |                                 result                            | : R2
01B3    796 ;       +----------------+----------------+----------------+----------------+
01B3    797 ;       |                                   dst                             | : R3
01B3    798 ;       +----------------+----------------+----------------+----------------+
01B3    799 ;
01B3    800 ;       Depending on where the exception occurred, some of these parameters
01B3    801 ;       may not be relevant. They are nevertheless stored as if they were
01B3    802 ;       valid to make this restart code as simple as possible.
01B3    803 ;
01B3    804 ;       R0<04:00> - Remaining digit/byte count in source string
01B3    805 ;       R0<07:05> - spare
01B3    806 ;       R0<15:08> - "srcaddr" difference (current - initial)
01B3    807 ;       R0<19:16> - Saved condition codes
01B3    808 ;       R0<22:20> - Restart code (identifies point where routine will resume)
01B3    809 ;       R0<23>    - Internal FPD flag
01B3    810 ;       R0<31:24> - Size of instruction in instruction stream (delta PC)
01B3    811 ;       R1        - Address of current byte in source string
01B3    812 ;       R2        - Value of intermediate or final result
01B3    813 ;       R3        - Address of destination (unchanged from input value of R3)
01B3    814 ;
01B3    815 ;       00(SP) - Return PC from VAX$CVTPL routine
01B3    816 ;
01B3    817 ; Implicit Input:
01B3    818 ;
01B3    819 ;       Note that the initial "srclen" is checked for legality before any
01B3    820 ;       restartable exception can occur. This means that R0 LEQU 31, which
01B3    821 ;       leaves bits <15:5> free for storing intermediate state. In the case of
01B3    822 ;       an access violation, R0<15:8> is used to store the difference between
01B3    823 ;       the original and current addresses in the source string.
01B3    824 ;
01B3    825 ; Output Parameters:
01B3    826 ;
01B3    827 ;       R0  - Updated digit or byte count in source string
01B3    828 ;       R2  - Condition codes reflecting result
01B3    829 ;       R3  - Address of destination (unchanged from input value)
01B3    830 ;       R6  - Intermediate (or final) longword result
01B3    831 ;       R10 - Address of CVTPL_ACCVIO, this module's "condition handler"
01B3    832 ;
01B3    833 ;       If the instruction was interrupted at mark point 6, the stack and R1
01B3    834 ;       contain different values than they do if the instruction was interrupted
```

```
                     ;       at any of the intermediate restart points.
                     ;
                     ;       Access violation occurred at restart points 1 through 5
                     ;
                     ;               R1 - Address of current byte in source string
                     ;
                     ;               00(SP) - Original value of R1 (scraddr)
                     ;               04(SP) - Saved R4
                     ;               08(SP) - Saved R5
                     ;               12(SP) - Saved R6
                     ;               16(SP) - Saved R7
                     ;               20(SP) - Saved R10
                     ;               24(SP) - Return PC from VAX$CVTPL routine
                     ;
                     ;       Access violation occurred at restart points 1 through 5
                     ;
                     ;               R1 - Address of most significant byte in source string
                     ;                    (original srcaddr)
                     ;
                     ;               00(SP) - Zero (will be restored to R2)
                     ;               04(SP) - Zero (will be restored to R3)
                     ;               08(SP) - Saved R4
                     ;               12(SP) - Saved R5
                     ;               16(SP) - Saved R6
                     ;               20(SP) - Saved R7
                     ;               24(SP) - Saved R10
                     ;               28(SP) - Return PC from VAX$CVTPL routine
                     ;
                     ; Implicit Output:
                     ;
                     ;       R4, R5, and R7 are used as scratch registers
                     ;-
                     ;
                     VAX$CVTPL_RESTART::
04F3 8F    BB        01B3    869          PUSHR   #^M<R0,R1,R4,R5,R6,R7,R10>      ; Save some registers
                     01B7    870          ESTABLISH_HANDLER       CVTPL_ACCVIO    ; Reload R10 with handler address
                     01BC    871
                     01BC    872 ; Make sure that the CVTPL_B_STATE byte is now on the stack (in R0 or R1)
                     01BC    873
                     01BC    874          ASSUME CVTPL_B_STATE LE 7
                     01BC    875
        04   EF       01BC    876          EXTZV   #CVTPL_V_STATE,-
        03               01BE    877                  #CVTPL_S_STATE,-
54   02 AE              01BF    878                  CVTPL_B_STATE(SP),R4            ; Put restart code into R4
                     01C2    879
                     01C2    880 ; The next two instructions reconstruct the initial value of "srcaddr" that
                     01C2    881 ; is stored on the stack just above the saved R4. This value will be loaded
                     01C2    882 ; into R1 when the instruction completes execution.
                     01C2    883
55   01 AE    9A     01C2    884          MOVZBL  CVTPL_B_DELTA_SRCADDR(SP),R5    ; Get the difference
04 AE    55   C2     01C6    885          SUBL2   R5,CVTPL_A_SRCADDR(SP)          ; Recreate the original R1
   50    50   9A     01CA    886          MOVZBL  R0,R0                           ; Clear out R0<31:8>
                     01CD    887
                     01CD    888 ; Make sure that the intermediate result is stored in R2
                     01CD    889
                     01CD    890          ASSUME CVTPL_L_RESULT EQ 8
                     01CD    891
```

```
                56   52   D0  01CD   892          MOVL    R2,P6
                     52   DC  01D0   893          MOVPSL  R2                              ; Get clean copy of PSL
                     00   EF  01D2   894          EXTZV   #CVTPL_V_SAVED_PSW,-            ; Retrieve saved copy of
                     04       01D4   895                  #CVTPL_S_SAVED_PSW,-            ;  condition codes
             55   02  AE       01D5   896                  CVTPL_B_STATE(SP),R5
                52   0F   8A  01D8   897          BICB2   #<PSL$M_N!PSL$M_Z!PSL$M_V!PSL$M_C>,R2   ; Clear condition codes
                52   55   88  01DB   898          BISB2   R5,R2                           ; Restore saved codes to R2
                             01DE   899
                             01DE   900 ; A check is made to determine whether the access violation occurred at
                             01DE   901 ; restart point number 6, where the stack is slightly different from its
                             01DE   902 ; state at the other exception points.
                             01DE   903
                5E   04   C0  01DE   904          ADDL    #4,SP                           ; Discard saved R0 place holder
                06   54   D1  01E1   905          CMPL    R4,#CVTPL_6_RESTART             ; Check for restart at the bitter end
                     05   1F  01E4   906          BLSSU   10$                             ; Branch if somewhere else
                51   8E   D0  01E6   907          MOVL    (SP)+,R1                        ; Restore saved "current" R1
                     7E   7C  01E9   908          CLRQ    -(SP)                           ; Store final values of R2 and R3
        54   FFFE'CF44   3C  01EB   909 10$:     MOVZWL  RESTART_PC_TABLE_BASE-2[R4],R4  ; Convert code to PC offset
             FE0A CF44   17  01F1   910          JMP     MODULE_BASE[R4]                 ; Get back to work
                             01F6   911
                             01F6   912          END_MARK_POINT          CVTPL_M_STATE
                             01F6   913
                             01F6   914          .END
```

```
...PC...                        = 000000C6                  VAX$REFLECT_FAULT               ********    X   00
...RESTART_PC...                = 000000C6                  VAX$REFLECT_TRAP                ********    X   00
...ROPRAND...                   = 0000001C R      02        VAX$ROPRAND                     ********    X   00
CVTPL_1                           00000160 R      02
CVTPL_1_RESTART                 = 00000001
CVTPL_2                           00000168 R      02
CVTPL_2_RESTART                 = 00000002
CVTPL_3                           0000016D R      02
CVTPL_3_RESTART                 = 00000003
CVTPL_4                           00000172 R      02
CVTPL_4_RESTART                 = 00000004
CVTPL_5                           00000177 R      02
CVTPL_5_RESTART                 = 00000005
CVTPL_6                           0000017C R      02
CVTPL_6_RESTART                 = 00000006
CVTPL_ACCVIO                      00000133 R      02
CVTPL_A_SRCADDR                 = 00000004
CVTPL_B_DELTA_PC                = 00000003
CVTPL_B_DELTA_SRCADDR           = 00000001
CVTPL_B_STATE                   = 00000002
CVTPL_L_RESULT                  = 00000008
CVTPL_M_FPD                     = 00000080
CVTPL_M_STATE                   = 00000070
CVTPL_RADRMOD                     00000124 R      02
CVTPL_S_SAVED_PSW               = 00000004
CVTPL_S_STATE                   = 00000003
CVTPL_V_FPD                     = 00000007
CVTPL_V_SAVED_PSW               = 00000000
CVTPL_V_STATE                   = 00000004
DECIMAL$BOUNDS_CHECK              ********    X   00
DECIMAL$PACKED_TO_BINARY_TABLE    ********    X   00
DECIMAL$STRIP_ZEROS_RO_RT         ********    X   00
DECIMAL_ROPRAND                   0000011B R      02
EXCEPTION_PSL                   = 0000002C
HANDLER_TABLE_BASE                00000000 R      04
INTEGER_OVERFLOW                  000000FD R      02
MODULE_BASE                     = 00000000 R      02
MODULE_END                      = 000001F6 R      02
PACK_L_SAVED_R1                 = 00000004
PACK_L_SAVED_R2                 = 00000008
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$V_IV                        = 00000005
PSL$V_V                         = 00000001
RESTART_PC_TABLE_BASE             00000000 R      05
RESTART_TABLE_SIZE              = 00000006
SRM$K_INT_OVF_T                 = 00000001
TABLE_SIZE                      = 00000006
VAX$CVTPL                         00000003 RG     02
VAX$CVTPL_RESTART                 000001B3 RG     02
VAX$EXIT_EMULATOR                 ********    X   00
VAX$RADRMOD                       ********    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	000001F6 (502.)	02 (2.)	PIC USR CON REL LCL SHR EXE RD NOWRT NOVEC LONG
PC_TABLE	0000000C (12.)	03 (3.)	PIC USR CON REL LCL SHR NOEXE RD NOWRT NOVEC BYTE
HANDLER_TABLE	0000000C (12.)	04 (4.)	PIC USR CON REL LCL SHR NOEXE RD NOWRT NOVEC BYTE
RESTART_PC_TABLE	0000000C (12.)	05 (5.)	PIC USR CON REL LCL SHR NOEXE RD NOWRT NOVEC BYTE

Performance indicators

Phase	Page faults	CPU Time	Elapsed Time
Initialization	15	00:00:00.02	00:00:02.22
Command processing	72	00:00:00.45	00:00:04.27
Pass 1	127	00:00:03.63	00:00:12.69
Symbol table sort	0	00:00:00.15	00:00:00.18
Pass 2	163	00:00:01.80	00:00:05.26
Symbol table output	7	00:00:00.08	00:00:00.66
Psect synopsis output	3	00:00:00.03	00:00:00.03
Cross-reference output	0	00:00:00.00	00:00:00.00
Assembler run totals	387	00:00:06.17	00:00:25.31

The working set limit was 1050 pages.
19456 bytes (38 pages) of virtual memory were used to buffer the intermediate code.
There were 10 pages of symbol table space allocated to hold 137 non-local and 27 local symbols.
914 source lines were read in Pass 1, producing 21 object records in Pass 2.
21 pages of virtual memory were used to define 19 macros.

Macro library statistics

Macro library name	Macros defined
_255DUA28:[EMULAT.OBJ]VAXMACROS.MLB;1	10
_255DUA28:[SYSLIB]STARLET.MLB;2	6
TOTALS (all libraries)	16

303 GETS were required to define 16 macros.

There were no errors, warnings or information messages.

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

VAXHANDLR
LIS

VAXCVTLP
LIS

VAXDECIML
LIS

VAXEMULAT
LIS

VAXCVTPL
LIS

VAXEDITPC
LIS