```
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  EEEEEEEEEE  DDDDDDDD     IIIIII    TTTTTTTTTT  PPPPPPPP      CCCCCCCC
VV      VV    AAAAAA    XX      XX  EEEEEEEEEE  DDDDDDDD     IIIIII    TTTTTTTTTT  PPPPPPPP      CCCCCCCC
VV      VV  AA      AA  XX      XX  EE          DD      DD     II        TT      PP      PP  CC
VV      VV  AA      AA  XX      XX  EE          DD      DD     II        TT      PP      PP  CC
VV      VV  AA      AA    XX  XX    EE          DD      DD     II        TT      PP      PP  CC
VV      VV  AA      AA    XX  XX    EE          DD      DD     II        TT      PP      PP  CC
VV      VV  AA      AA      XX      EEEEEEEE    DD      DD     II        TT      PPPPPPPP    CC
VV      VV  AA      AA      XX      EEEEEEEE    DD      DD     II        TT      PPPPPPPP    CC
VV      VV  AAAAAAAAAA    XX  XX    EE          DD      DD     II        TT      PP          CC
VV      VV  AAAAAAAAAA    XX  XX    EE          DD      DD     II        TT      PP          CC
  VV  VV    AA      AA  XX      XX  EE          DD      DD     II        TT      PP          CC              ....
  VV  VV    AA      AA  XX      XX  EE          DD      DD     II        TT      PP          CC              ....
    VV      AA      AA  XX      XX  EEEEEEEEEE  DDDDDDDD     III'III   TT      PP            CCCCCCCC      ....
    VV      AA      AA  XX      XX  EEEEEEEEEE  DDDDDDDD     II'III    TT      PP            CCCCCCCC      ....

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
```

VAX
V04

Table of contents

```
0000     1          .TITLE  VAX$EDITPC - VAX-11 EDITPC Instruction Emulation
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 ;
0000    36 ;       The routines in this module emulate the VAX-11 EDITPC instruction.
0000    37 ;       These routines can be a part of an emulator package or can be
0000    38 ;       called directly after the input parameters have been loaded into
0000    39 ;       the architectural registers.
0000    40 ;
0000    41 ;       The input parameters to these routines are the registers that
0000    42 ;       contain the intermediate instruction state.
0000    43 ;
0000    44 ; Environment:
0000    45 ;
0000    46 ;       These routines run at any access mode, at any IPL, and are AST
0000    47 ;       reentrant.
0000    48 ;
0000    49 ; Author:
0000    50 ;
0000    51 ;       Lawrence J. Kenah
0000    52 ;
0000    53 ; Creation Date
0000    54 ;
0000    55 ;       20 September 1982
0000    56 ;
0000    57 ; Modified by:
```

```
0000     58 ;
0000     59 ;       V01-008 LJK0035         Lawrence J. Kenah       16-Jul-1984
0000     60 ;               Fix bugs in restart logic.
0000     61 ;
0000     62 ;               R6 cannot be used as both the exception dispatch register and
0000     63 ;                 a scratch register in the main EDITPC routine. Use R7 as the
0000     64 ;                 scratch register.
0000     65 ;               Add code to the EDITPC_1 restart routine to restore R7 as the
0000     66 ;                 address of the sign byte.
0000     67 ;               Clear C-bit in saved PSW in END_FLOAT_1 routine.
0000     68 ;               Restore R9 (count of zeros) with CVTWL instruction.
0000     69 ;               Fix calculation of initial srcaddr parameter.
0000     70 ;               Preserve R8 in READ_1 and READ_2 routines.
0000     71 ;               Preserve R7 in FLOAT_2 routine.
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. Load STATE
0000     76 ;               cell with nonzero restart code in ROPRAND_FAULT routine.
0000     77 ;
0000     78 ;       V01-006 LJK0026         Lawrence J. Kenah       19-Mar-1984
0000     79 ;               Final cleanup, especially in access violation handling. Make
0000     80 ;               all of the comments in exception handling accurately describe
0000     81 ;               what the code is really doing.
0000     82 ;
0000     83 ;       V01-005 LJK0018         Lawrence J. Kenah       23-Jan-1984
0000     84 ;               Add restart logic for illegal pattern operator. Add access
0000     85 ;               violation handling.
0000     86 ;
0000     87 ;       V01-004 LJK0014         Lawrence J. Kenah       21-Nov-1983
0000     88 ;               Clean up rest of exception handling. Remove reference
0000     89 ;               to LIB$SIGNAL.
0000     90 ;
0000     91 ;       V01-003 LJK0012         Lawrence J. Kenah       8-Nov-1983
0000     92 ;               Start out with R9 containing zero so that pattern streams
0000     93 ;               that do not contain EO$ADJUST_INPUT will work correctly.
0000     94 ;
0000     95 ;       V01-002 LJK0009         Lawrence J. Kenah       20-Oct-1983
0000     96 ;               Add exception handling. Fix bug in size of count field.
0000     97 ;
0000     98 ;       V01-001 Original        Lawrence J. Kenah       20-Sep-1982
0000     99 ;--
```

```
                     0000   101          .SUBTITLE       Declarations
                     0000   102
                     0000   103 ; Include files
                     0000   104
                     0000   105          .NOCROSS                        ; No cross reference for these
                     0000   106          .ENABLE         SUPPRESSION     ; No symbol table entries either
                     0000   107
                     0000   108          EDITPC_DEF                      ; Define intermediaie instruction state
                     0000   109
                     0000   110          PACK_DEF                        ; Stack offsets for exceptions
                     0000   111
                     0000   112          $PSLDEF                         ; Define bit fields in PSL
                     0000   113
                     0000   114          .DISABLE        SUPPRESSION     ; Turn on symbol table again
                     0000   115          .CROSS                          ; Cross reference is OK now
                     0000   116
                     0000   117 ; Equated symbols
                     0000   118
           00000020  0000   119          BLANK = ^A' '
           0000002D  0000   120          MINUS = ^A'-'
           00000030  0000   121          ZERO  = ^A'0'
                     0000   122
                     0000   123 ; Local macro definitions
                     0000   124
                     0000   125          .MACRO  EO_READ
                     0000   126          RESTART_POINT
                     0000   127          BSBW    EO_READ
                     0000   128          .ENDM   EO_READ
                     0000   129
                     0000   130 ; External declarations
                     0000   131
                     0000   132          .DISABLE        GLOBAL
                     0000   133
                     0000   134          .EXTERNAL -
                     0000   135                          VAX$REFLECT_FAULT,-
                     0000   136                          VAX$ROPRAND,-
                     0000   137                          VAX$EDITPC_OVERFLOW
                     0000   138 ; PSECT Declarations:
                     0000   139
                     0000   140          .DEFAULT        DISPLACEMENT , WORD
                     0000   141
                 00000000   142          .PSECT _VAX$CODE PIC, USR, CON, REL, LCL, SHR, EXE, RD, NOWRT, LONG
                     0000   143
                     0000   144          BEGIN_MARK_POINT        RESTART
```

```
                        .SUBTITLE       VAX$EDITPC - Edit Packed to Character String
                ;+
                ; Functional Description:
                ;
                ;       The destination string specified by the pattern and destination
                ;       address operands is replaced by the editted version of the source
                ;       string specified by the source length and source address operands.
                ;       The editing is performed according to the pattern string starting at
                ;       the address pattern and extending until a pattern end (EO$END) pattern
                ;       operator is encountered.  The pattern string consists of one byte
                ;       pattern operators.  Some pattern operators take no operands.  Some
                ;       take a repeat count which is contained in the rightmost nibble of the
                ;       pattern operator itself.  The rest take a one byte operand which
                ;       follows the pattern operator immediately.  This operand is either an
                ;       unsigned integer length or a byte character.  The individual pattern
                ;       operators are described on the following pages.
                ;
                ; Input Parameters:
                ;
                ;       R0 - srclen.rw          Length of input packed decimal string
                ;       R1 - srcaddr.ab         Address of input packed decimal string
                ;       R3 - pattern.ab         Address of table of editing pattern operators
                ;       R5 - dstaddr.ab         Address of output character string
                ;
                ; Intermediate State:
                ;
                ;    31              23              15              07              00
                ;    +---------------+---------------+---------------+---------------+
                ;    |          zero count           |             srclen            | : R0
                ;    +---------------+---------------+---------------+---------------+
                ;    |                             srcaddr                           | : R1
                ;    +---------------+---------------+---------------+---------------+
                ;    | delta-srcaddr |    delta-PC   |      sign     |      fill     | : R2
                ;    +---------------+---------------+---------------+---------------+
                ;    |                             pattern                           | : R3
                ;    +---------------+---------------+---------------+---------------+
                ;    |   loop-count  |     state     |   saved-PSW   |   inisrclen   | : R4
                ;    +---------------+---------------+---------------+---------------+
                ;    |                             dstaddr                           | : R5
                ;    +---------------+---------------+---------------+---------------+
                ;
                ; Output Parameters:
                ;
                ;       R0 - Length of input decimal string
                ;       R1 - Address of most significant byte of input decimal string
                ;       R2 - 0
                ;       R3 - Address of byte containing EO$END pattern operator
                ;       R4 - 0
                ;       R5 - Address of one byte beyond destination character string
                ;
                ; Condition Codes:
                ;
                ;       N <- source string LSS 0        (src = -0 => N = 0)
                ;       Z <- source string EQL 0
                ;       V <- decimal overflow           (nonzero digits lost)
                ;       C <- significance
                ;-
```

```
                                0000   203
                                0000   204          .ENABLE LOCAL_BLOCK
                                0000   205
                                0000   206          ASSUME EDITPC_B_STATE EQ 18     ; Make sure we test the right FPD bit
                                0000   207
                     02DD   31  0000   208 2$:      BRW     VAX$EDITPC_RESTART      ; Restart somewhere else
                                0003   209
                     01EE   31  0003   210 5$:      BRW     EDITPC_ROPRAND_ABORT    ; Time to quit if illegal length
                                0006   211
                                0006   212 VAX$EDITPC::
                F6 54    14    E0  0006   213          BBS     #<EDITPC_V_FPD+16>,R4,2$        ; Branch if this is a restart
                   0FC3  8F    BB  000A   214          PUSHR   #^M<R0,R1,R6,R7,R8,R9,R10,R11>  ; Save lots of registers
                   1F    50    B1  000E   215          CMPW    R0,#31                  ; Check for R0 GTRU 31
                         F0    1A  0011   216          BGTRU   5$                      ; Signal ROPRAND if R0 GTRU 31
                   50    50    3C  0013   217          MOVZWL  R0,R0                   ; Clear any junk from high-order word
                   52    20    9A  0016   218          MOVZBL  #BLANK,R2               ; Set fill to BLANK, stored in R2
                         59    D4  0019   219          CLRL    R9                      ; Start with "zero count" of zero
                                   001B   220          ESTABLISH_HANDLER       EDITPC_ACCVIO
                         5B    DC  0020   221          MOVPSL  R11                     ; Get current PSL
                   5B    0B    8A  0022   222          BICB    #<PSL$M_N!PSL$M_V!PSL$M_C>,R11  ; Clear N-, V-, and C-bits
                   5B    04    88  0025   223          BISB    #PSL$M_Z,R11            ; Set Z-bit.
                                0028   224
                                0028   225 ; We need to determine the sign in the input decimal string to choose
                                0028   226 ; the initial setting of the N-bit in the saved PSW.
                                0028   227
          57    50    04    01  EF  0028   228          EXTZV   #1,#4,R0,R7             ; Get byte offset to end of string
                      57    51  C0  002D   229          ADDL    R1,R7                   ; Get address of byte containing sign
                                0030   230          MARK_POINT      EDITPC_1 , RESTART
          57    67    04    00  EF  0030   231          EXTZV   #0,#4,(R7),R7           ; Get sign "digit" into R7
                                0035   232
                                0035   233          CASE    R7,LIMIT=#10,TYPE=B,<-  ; Dispatch on sign
                                0035   234                  20$,-                   ; 10 => +
                                0035   235                  10$,-                   ; 11 => -
                                0035   236                  20$,-                   ; 12 => +
                                0035   237                  10$,-                   ; 13 => -
                                0035   238                  20$,-                   ; 14 => +
                                0035   239                  20$,-                   ; 15 => +
                                0035   240                  >
                                0045   241
                                0045   242 ; Sign is MINUS
                                0045   243
                   5B    08    88  0045   244 10$:     BISB    #PSL$M_N,R11            ; Set N-bit in saved PSW
                   54    2D    9A  0048   245          MOVZBL  #MINUS,R4               ; Set sign to MINUS, stored in R4
                         03    11  004B   246          BRB     TOP_OF_LOOP             ; Join common code
                                004D   247
                                004D   248 ; Sign is PLUS (but initial content of sign register is BLANK)
                                004D   249
                   54    20    9A  004D   250 20$:     MOVZBL  #BLANK,R4               ; Set sign to BLANK, stored in R4
                                0050   251
                                0050   252 ; The architectural description of the EDITPC instruction uses an exit flag
                                0050   253 ; to determine whether to continue reading edit operators from the input
                                0050   254 ; stream. This implementation does not use an explicit exit flag. Rather, all
                                0050   255 ; of the end processing is contained in the routine that handles the EO$END
                                0050   256 ; operator.
                                0050   257
                                0050   258 ; The next several instructions are the main routine in this module. Each
                                0050   259 ; pattern is used to dispatch to a pattern-specific routine that performs
```

```
                     0050   260 ; its designated action. These routines (except for EO$END) return control
                     0050   261 ; to TOP_OF_LOOP to allow the next pattern operator to be processed.
                     0050   262
                     0050   263 TOP_OF_LOOP:
FD AF    9F          0050   264         PUSHAB  B^TOP_OF_LOOP                   ; Store "return PC"
                     0053   265
                     0053   266 ; The following instructions pick up the next byte in the pattern stream and
                     0053   267 ; dispatch to a pattern specific subroutine that performs the designated
                     0053   268 ; action. Control is passed back to the main EDITPC loop by the RSB
                     0053   269 ; instructions located in each pattern-specific subroutine.
                     0053   270
                     0053   271 ; Note that the seemingly infinite loop actually terminates when the EO$END
                     0053   272 ; pattern operator is detected. That routine insures that we do not return
                     0053   273 ; to this loop but rather to the caller of VAX$EDITPC.
                     0053   274
                     0053   275         MARK_POINT      EDITPC_2 , RESTART
                     0053   276         CASE    (R3)+,LIMIT=#0,TYPE=B,<-
                     0053   277                 EO$END_ROUTINE,-                ; 00 - EO$END
                     0053   278                 EO$END_FLOAT_ROUTINE,-          ; 01 - EO$END_FLOAT
                     0053   279                 EO$CLEAR_SIGNIF_ROUTINE,-       ; 02 - EO$CLEAR_SIGNIF
                     0053   280                 EO$SET_SIGNIF_ROUTINE,-         ; 03 - EO$SET_SIGNIF
                     0053   281                 EO$STORE_SIGN_ROUTINE,-         ; 04 - EO$STORE_SIGN
                     0053   282                 >
                     0061   283
                     0061   284         MARK_POINT      EDITPC_3
                     0061   285         CASE    -1(R3),LIMIT=#^X40,TYPE=B,<-
                     0061   286                 EO$LOAD_FILL_ROUTINE,-          ; 40 - EO$LOAD_FILL
                     0061   287                 EO$LOAD_SIGN_ROUTINE,-          ; 41 - EO$LOAD_SIGN
                     0061   288                 EO$LOAD_PLUS_ROUTINE,-          ; 42 - EO$LOAD_PLUS
                     0061   289                 EO$LOAD_MINUS_ROUTINE,-         ; 43 - EO$LOAD_MINUS
                     0061   290                 EO$INSERT_ROUTINE,-             ; 44 - EO$INSERT
                     0061   291                 EO$BLANK_ZERO_ROUTINE,-         ; 45 - EO$BLANK_ZERO
                     0061   292                 EO$REPLACE_SIGN_ROUTINE,-       ; 46 - EO$REPLACE_SIGN
                     0061   293                 EO$ADJUST_INPUT_ROUTINE,-       ; 47 - EO$ADJUST_INPUT
                     0061   294                 >
                     0077   295
                     0077   296         MARK_POINT      EDITPC_4
          FF A3   0F 93    0077   297         BITB    #^B1111,-1(R3)          ; Check for 80, 90, or A0
                  10 13    007B   298         BEQL    30$                     ; Reserved operand on repeat of zero
                     007D   299         MARK_POINT      EDITPC_5
57 FF A3  04 04    EF    007D   300         EXTZV   #4,#4,-1(R3),R7         ; Ignore repeat count in dispatch
                     0083   301
                     0083   302         CASE    R7,LIMIT=#8,TYPE=B,<-
                     0083   303                 EO$FILL_ROUTINE,-               ; 81 to 8F - EO$FILL
                     0083   304                 EO$MOVE_ROUTINE,-               ; 91 to 9F - EO$MOVE
                     0083   305                 EO$FLOAT_ROUTINE,-              ; A1 to AF - EO$FLOAT
                     0083   306                 >
                     008D   307
                     008D   308 ; If we drop through all three CASE instructions, the pattern operator is
                     008D   309 ; unimplemented or reserved. R3 is backed up to point to the illegal
                     008D   310 ; pattern operator and a reserved operand FAULT is signalled.
                     008D   311
                  53 D7      008D   312 30$:    DECL    R3                      ; Point R3 to illegal operator
               5E 04 C0      008F   313         ADDL    #4,SP                   ; Discard return PC
                  011E 31    0092   314         BRW     EDITPC_ROPRAND_FAULT    ; Initiate exception processing
                     0095   315
                     0095   316         .DISABLE        LOCAL_BLOCK
```

```
0095    318          .SUBTITLE       Description of Pattern-Specific Routines
0095    319 ;+
0095    320 ; Functional Description:
0095    321 ;
0095    322 ;       There is a separate action routine for each pattern operator. These
0095    323 ;       routines are entered with specific register contents and several
0095    324 ;       scratch registers at their disposal. They perform their designated
0095    325 ;       action and return to the main VAX$EDITPC routine.
0095    326 ;
0095    327 ;       There are several words used in the architectural description of this
0095    328 ;       instruction that are carried over into comments in this module. These
0095    329 ;       words are briefly mentioned here.
0095    330 ;
0095    331 ;       char    Character in byte following pattern operator (used by
0095    332 ;               EO$LOAD_FILL, EO$LOAD_SIGN, EO$LOAD_PLUS, EO$LOAD_MINUS,
0095    333 ;               and EO$INSERT)
0095    334 ;
0095    335 ;       length  Length in byte following pattern operator (used by
0095    336 ;               EO$BLANK_ZERO, EO$REPLACE_SIGN, and EO$ADJUST_INPUT)
0095    337 ;
0095    338 ;       repeat  Repeat count in bits <3:0> of pattern operator (used by
0095    339 ;               EO$FILL, EO$MOVE, and EO$FLOAT)
0095    340 ;
0095    341 ;       The architecture makes use of two character registers, described
0095    342 ;       as appearing in different bytes of R2. For simplicity, we use an
0095    343 ;       additional register.
0095    344 ;
0095    345 ;       fill    Stored in R2<7:0>
0095    346 ;
0095    347 ;       sign    Stored in R4<7:0>
0095    348 ;
0095    349 ;       Finally, the architecture describes two subroutines, one that obtains
0095    350 ;       the next digit from the input string and the other that stores a
0095    351 ;       character in the output string.
0095    352 ;
0095    353 ;       READ    Subroutine EO_READ provides this functionality
0095    354 ;
0095    355 ;       STORE   A single instruction of the form
0095    356 ;
0095    357 ;                       MOVB    xxx,(R5)+
0095    358 ;
0095    359 ;               or
0095    360 ;
0095    361 ;                       ADDB3   #ZERO,R7,(R5)+
0095    362 ;
0095    363 ;               stores a single character and advances the pointer.
0095    364 ;
0095    365 ; Input Parameters:
0095    366 ;
0095    367 ;       R0 - Updated length of input decimal string
0095    368 ;       R1 - Address of next byte of input decimal string
0095    369 ;       R2 - Fill character
0095    370 ;       R3 - Address of one byte beyond current pattern operator
0095    371 ;       R4 - Sign character
0095    372 ;       R5 - Address of next character to be stored in output character string
0095    373 ;
0095    374 ; Implicit Input:
```

```
0095   375 ;
0095   376 ;       Several registers are used to contain intermediate state, passed
0095   377 ;       from one action routine to the next.
0095   378 ;
0095   379 ;       R7  - Contains latest digit from input stream (output from EO_READ)
0095   380 ;       R8  - Used as loop counter
0095   381 ;       R9  - Contains the value described in the architecture as R0<31:16>
0095   382 ;       R11 - Pseudo-PSW that contains the saved condition codes
0095   383 ;
0095   384 ; Side Effects:
0095   385 ;
0095   386 ;       The remaining registers are used as scratch by the action routines.
0095   387 ;
0095   388 ;       R6 - Scratch register used only by access violation handler
0095   389 ;       R7 - Output parameter of EO_READ routine
0095   390 ;       R8 - Scratch register used by pattern-specific routines
0095   391 ;
0095   392 ; Output Parameters:
0095   393 ;
0095   394 ;       The actual output depends on the pattern operator that is currently
0095   395 ;       executing. The routine headers for each routine will describe the
0095   396 ;       specific output parameters.
0095   397 ;-
```

```
0095   399          .SUBTITLE       Utility Subroutine (READ Next Digit)
0095   400 ;+
0095   401 ; Functional Description:
0095   402 ;
0095   403 ;       This routine reads the next digit from the input packed decimal
0095   404 ;       string and passes it back to the caller.
0095   405 ;
0095   406 ; Input Parameters:
0095   407 ;
0095   408 ;       R0 - Updated length of input decimal string
0095   409 ;       R1 - Address of next byte of input decimal str ng
0095   410 ;       R9 - Count of extra zeros (see EO$ADJUST_INPUT)
0095   411 ;
0095   412 ;       (SP) - Return address to caller of this routine
0095   413 ;
0095   414 ;       Note that R9<15:0> contains the data described by the architecture as
0095   415 ;       appearing in R0<31:16>. In the event of an restartable exception
0095   416 ;       (access violation or reserved operand fault due to an illegal pattern
0095   417 ;       operator), the contents of R9<15:0> will be stored in R0<31:16>. In
0095   418 ;       order for the instruction to be restarted, the "zero count" (the
0095   419 ;       contents of R9) must be preserved. While any available field will do
0095   420 ;       in the event of an access violation, the use of R0<31:16> is clearly
0095   421 ;       specified for a reserved operand fault.
0095   422 ;
0095   423 ; Output Parameters:
0095   424 ;
0095   425 ;       The behavior of this routine depends on the contents of R9
0095   426 ;
0095   427 ;       R9 is zero on input
0095   428 ;
0095   429 ;               R0 - Updated by one
0095   430 ;               R1 - Updated by one if R0<0> is clear on input
0095   431 ;               R7 - Next decimal digit in input string
0095   432 ;               R9 - Unchanged
0095   433 ;
0095   434 ;               PSW<Z> is set if the digit is zero, clear otherwise
0095   435 ;
0095   436 ;       R9 is nonzero (LSS 0) on input
0095   437 ;
0095   438 ;               R0 - Unchanged
0095   439 ;               R1 - Unchanged
0095   440 ;               R7 - Zero
0095   441 ;               R9 - Incremented by one (toward zero)
0095   442 ;
0095   443 ;               PSW<Z> is set
0095   444 ;-
0095   445
0095   446 EO_READ:
   59   D5  0095   447          TSTL    R9                      ; Check for "R0" LSS 0
   13   12  0097   448          BNEQ    20$                     ; Special code if nonzero
   50   D7  0099   449          DECL    R0                      ; Insure that digits still remain
   14   19  009B   450          BLSS    30$                     ; Reserved operand if none
06 50   E9  009D   451          BLBC    R0,10$                  ; Next code path is flip flop
               00A0   452
               00A0   453 ; R0 was even on input (and is now odd), indicating that we want the low
               00A0   454 ; order nibble in the input stream. The input pointer R1 must be advanced
               00A0   455 ; to point to the next byte.
```

```
                                00A0    456
                                00A0    457          MARK_POINT      READ_1
            57   81   04   00  EF 00A0    458          EXTZV   #0,#4,(R1)+,R7          ; Load low order nibble into R7
                             05 00A5    459          RSB                             ; Return with information in Z-bit
                                00A6    460
                                00A6    461 ; R0 was odd on input (and is now even), indicating that we want the high
                                00A6    462 ; order nibble in the input stream. The next pass through this routine will
                                00A6    463 ; pick up the low order nibble of the same input byte.
                                00A6    464
                                00A6    465          MARK_POINT      READ_2
            57   61   04   04  EF 00A6    466 10$:     EXTZV   #4,#4,(R1),R7           ; Load high order nibble into R7
                             05 00AB    467          RSB                             ; Return with information in Z-bit
                                00AC    468
                                00AC    469 ; R9 was nonzero on input, indicating that zeros should replace the original
                                00AC    470 ; input digits.
                                00AC    471
                      59    D6 00AC    472 20$:     INCL    R9                      ; Advance R9 toward zero
                      57    D4 00AE    473          CLRL    R7                      ; Behave as if we read a zero digit
                             05 00B0    474          RSB                             ; Return with Z-bit set
                                00B1    475
                                00B1    476 ; The input decimal string ran out of digits before its time. The architecture
                                00B1    477 ; dictates that R3 points to the pattern operator that requested the input
                                00B1    478 ; digit and R0 contains a -1 when the reserved operand abort is reported.
                                00B1    479 ; It is not necessary to load R0 here. R0 already contains -1 because it just
                                00B1    480 ; turned negative.
                                00B1    481
                      53    D7 00B1    482 30$:     DECL    R3                      ; Back up R3 to current pattern operator
                5E    08    C0 00B3    483          ADDL    #8,SP                   ; Discard two return PCs
                   013B     31 00B6    484          BRW     EDITPC_ROPRAND_ABORT    ; Branch aid for reserved operand abort
```

```
                                00B9   486          .SUBTITLE       EO$INSERT - Insert Character
                                00B9   487 ;+
                                00B9   488 ; Functional Description:
                                00B9   489 ;
                                00B9   490 ;       Insert a fixed character, substituting the fill character if
                                00B9   491 ;       not significant.
                                00B9   492 ;
                                00B9   493 ; Input Parameters:
                                00B9   494 ;
                                00B9   495 ;       R2 - Fill character
                                00B9   496 ;       R3 - Address of character to be inserted if significance is set
                                00B9   497 ;       R5 - Address of next character to be stored in output character string
                                00B9   498 ;       R11<C> - Current setting of significance
                                00B9   499 ;
                                00B9   500 ; Output Parameters:
                                00B9   501 ;
                                00B9   502 ;       Character in pattern stream (or fill character if no significance)
                                00B9   503 ;       is stored in the the output string.
                                00B9   504 ;
                                00B9   505 ;       R3 - Advanced beyond character in pattern stream
                                00B9   506 ;       R5 - Advanced one byte as a result of the STORE operation
                                00B9   507 ;-
                                00B9   508
                                00B9   509 EO$INSERT_ROUTINE:
                  04 5B   00    E1  00B9   510          BBC     #PSL$V_C,R11,10$        ; Skip next if no significance
                                00BD   511          MARK_POINT      INSERT_1
                     85   83    90  00BD   512          MOVB    (R3)+,(R5)+             ; STORE "ch" in output string
                               05   00C0   513          RSB
                                00C1   514
                                00C1   515          MARK_POINT      INSERT_2
                  85   52    90  00C1   516 10$:     MOVB    R2,(R5)+                ; STORE fill character
                       53    D6  00C4   517          INCL    R3                      ; Skip over unused character
                               05   00C6   518          RSB
```

```
                     00C7   520          .SUBTITLE       EO$STORE_SIGN - Store Sign
                     00C7   521 ;+
                     00C7   522 ; Functional Description:
                     00C7   523 ;
                     00C7   524 ;       The contents of the sign register are placed into the output string.
                     00C7   525 ;
                     00C7   526 ; Input Parameters:
                     00C7   527 ;
                     00C7   528 ;       R4 - Sign character
                     00C7   529 ;       R5 - Address of next character to be stored in output character string
                     00C7   530 ;
                     00C7   531 ; Output Parameters:
                     00C7   532 ;
                     00C7   533 ;       Sign character is stored in the the output string.
                     00C7   534 ;
                     00C7   535 ;       R5 - Advanced one byte as a result of the STORE operation
                     00C7   536 ;-
                     00C7   537
                     00C7   538 EO$STORE_SIGN_ROUTINE:
                     00C7   539          MARK_POINT      STORE_SIGN_1
85   54   90         00C7   540          MOVB    R4,(R5)+                ; STORE sign character
     05              00CA   541          RSB
```

```
                                                     M 8
                                00CB    543          .SUBTITLE       EO$FILL - Store Fill
                                00CB    544 ;+
                                00CB    545 ; Functional Description:
                                00CB    546 ;
                                00CB    547 ;       The contents of the fill register are placed into the output string
                                00CB    548 ;       a total of "repeat" times.
                                00CB    549 ;
                                00CB    550 ; Input Parameters:
                                00CB    551 ;
                                00CB    552 ;       R2 - Fill character
                                00CB    553 ;       R5 - Address of next character to be stored in output character string
                                00CB    554 ;
                                00CB    555 ;       -1(R3)<3:0> - Repeat count is stored in right nibble of pattern operator
                                00CB    556 ;
                                00CB    557 ; Output Parameters:
                                00CB    558 ;
                                00CB    559 ;       Fill character is stored in the output string "repeat" times
                                00CB    560 ;
                                00CB    561 ;       R5 - Advanced "repeat" bytes as a result of the STORE operations
                                00CB    562 ;-
                                00CB    563
                                00CB    564 EO$FILL_ROUTINE:
                                00CB    565          MARK_POINT      FILL_1
     58    FF A3    04    00    EF  00CB    566          EXTZV   #0,#4,-1(R3),R8         ; Get repeat count from pattern operator
                                00D1    567          MARK_POINT      FILL_2 , RESTART
                   85    52    90  00D1    568 10$:     MOVB    R2,(R5)+                ; STORE fill character
                     FA 58    F5  00D4    569          SOBGTR  R8,10$                  ; Test for end of loop
                              05  00D7    570          RSB
```

```
                                00D8   572          .SUBTITLE       EO$MOVE - Move Digits
                                00D8   573 ;+
                                00D8   574 ; Functional Description:
                                00D8   575 ;
                                00D8   576 ;       The right nibble of the pattern operator is  the  repeat  count.   For
                                00D8   577 ;       repeat  times, the following algorithm is executed.  The next digit is
                                00D8   578 ;       moved from the source to the destination.  If the digit  is  non-zero,
                                00D8   579 ;       significance  is  set  and  zero  is  cleared.   If  the  digit is not
                                00D8   580 ;       significant (i.e., is a leading zero) it is replaced by  the  contents
                                00D8   581 ;       of the fill register in the destination.
                                00D8   582 ;-
                                00D8   583
                                00D8   584 EO$MOVE_ROUTINE:
                                00D8   585          MARK_POINT      MOVE_1
    58    FF A3    04    00   EF  00D8   586          EXTZV   #0,#4,-1(R3),R8         ; Get repeat count
                                00DE   587
                                00DE   588 10$:     EO_READ                         ; Get next input digit
                          0E    13  00E1   589          BEQL    30$                     ; Is it zero? Branch if yes
                       5B 01    88  00E3   590          BISB    #PSL$M_C,R11            ; Indicate significance
                       5B 04    8A  00E6   591          BICB    #PSL$M_Z,R11            ; Also indicate nonzero
                                00E9   592
                                00E9   593          MARK_POINT      MOVE_2 , RESTART
              85    57   30    81  00E9   594 20$:     ADDB3   #ZERO,R7,(R5)+          ; STORE digit in output stream
                EE 58    F5      00ED   595          SOBGTR  R8,10$                  ; Test for end of loop
                         05      00F0   596          RSB
                                00F1   597
            F4 5B    00    E0  00F1   598 30$:     BBS     #PSL$V_C,R11,20$        ; If significance, then STORE digit
                                00F5   599
                                00F5   600          MARK_POINT      MOVE_3 , RESTART
                85   52    90  00F5   601          MOVB    R2,(R5)+                ; Otherwise, STORE fill character
                  E3 58    F5      00F8   602          SOBGTR  R8,10$                  ; Test for end of loop
                         05      00FB   603          RSB
```

```
                        00FC    605          .SUBTITLE       EO$FLOAT - Float Sign
                        00FC    606 ;+
                        00FC    607 ; Functional Description:
                        00FC    608 ;
                        00FC    609 ;       The right nibble of the pattern operator is  the  repeat  count.   For
                        00FC    610 ;       repeat  times,  the  following  algorithm is executed.  The next digit
                        00FC    611 ;       from the source is examined.  If it is non-zero  and  significance  is
                        00FC    612 ;       not  yet  set, then the contents of the sign register is stored in the
                        00FC    613 ;       destination, significance is set, and zero is cleared.  If  the  digit
                        00FC    614 ;       is  significant,  it  is  stored  in  the  destination,  otherwise the
                        00FC    615 ;       contents of the fill register is stored in the destination.
                        00FC    616 ;-
                        00FC    617
                        00FC    618 EO$FLOAT_ROUTINE:
                        00FC    619          MARK_POINT      FLOAT_1
 58   FF A3   04   00   EF  00FC    620          EXTZV   #0,#4,-1(R3),R8         ; Get repeat count
                        0102    621
                        0102    622 10$:     EO_READ                         ; Get next input digit
          0B 5B   00   E0  0105    623          BBS     #PSL$V_C,R11,20$        ; Is significance set? Branch if yes.
                     11   13  0109    624          BEQL    30$                     ; Is digit zero? Branch if yes.
                        010B    625          MARK_POINT      FLOAT_2 , RESTART
             85   54   90  010B    626          MOVB    R4,(R5)+                ; STORE sign
             5B   01   88  010E    627          BISB    #PSL$M_C,R11            ; Indicate significance
             5B   04   8A  0111    628          BICB    #PSL$M_Z,R11            ; Also indicate nonzero
                        0114    629
                        0114    630          MARK_POINT      FLOAT_3 , RESTART
        85   57   30   81  0114    631 20$:     ADDB3   #ZERO,R7,(R5)+          ; STORE digit in output stream
               E7 58   F5  0118    632          SOBGTR  R8,10$                  ; Test for end of loop
                     05   011B    633          RSB
                        011C    634
                        011C    635 30$:     MARK_POINT      FLOAT_4 , RESTART
             85   52   90  011C    636          MOVB    R2,(R5)+                ; Otherwise, STORE fill character
               E0 58   F5  011F    637          SOBGTR  R8,10$                  ; Test for end of loop
                     05   0122    638          RSB
```

```
                     0123   640          .SUBTITLE       E0$END_FLOAT - End Floating Sign
                     0123   641 ;+
                     0123   642 ; Functional Description:
                     0123   643 ;
                     0123   644 ;       If the floating sign has not yet been placed into the destination
                     0123   645 ;       string (that is, if significance is not yet set), then the contents
                     0123   646 ;       of the sign register are stored in the output string and significance
                     0123   647 ;       is set.
                     0123   648 ;
                     0123   649 ; Input Parameters:
                     0123   650 ;
                     0123   651 ;       R4 - Sign character
                     0123   652 ;       R5 - Address of next character to be stored in output character string
                     0123   653 ;       R11<C> - Current setting of significance
                     0123   654 ;
                     0123   655 ; Output Parameters:
                     0123   656 ;
                     0123   657 ;       Sign character is optionally stored in the output string (if
                     0123   658 ;       significance was not yet set).
                     0123   659 ;
                     0123   660 ;       R5 - Optionally advanced one byte as a result of the STORE operation
                     0123   661 ;       R11<C> - (Significance) is unconditionally SET
                     0123   662 ;-
                     0123   663
                     0123   664 E0$END_FLOAT_ROUTINE:
03 5B    00    E2    0123   665          BBSS    #PSL$V_C,R11,10$        ; Test and set significance
                     0127   666          MARK_POINT      END_FLOAT_1
   85    54    90    0127   667          MOVB    R4,(R5)+                ; STORE sign character
                05    012A   668 10$:     RSB
```

```
                     012B   670          .SUBTITLE       EO$BLANK_ZERO - Blank Backwards When Zero
                     012B   671 ;+
                     012B   672 ; Functional Description:
                     012B   673 ;
                     012B   674 ;       The pattern operator is followed by an unsigned byte  integer  length.
                     012B   675 ;       If  the  value  of the source string is zero, then the contents of the
                     012B   676 ;       fill register are stored into the last length bytes of the destination
                     012B   677 ;       string.
                     012B   678 ;
                     012B   679 ; Input Parameters:
                     012B   680 ;
                     012B   681 ;       R2 - Fill character
                     012B   682 ;       R3 - Address of "length", number of characters to blank
                     012B   683 ;       R5 - Address of next character to be stored in output character string
                     012B   684 ;       R11<Z> - Set if input string is zero
                     012B   685 ;
                     012B   686 ; Output Parameters:
                     012B   687 ;
                     012B   688 ;       Contents of fill register are stored in last "length" characters
                     012B   689 ;       of output string if input string is zero.
                     012B   690 ;
                     012B   691 ;       R3 - Advanced one byte over "length"
                     012B   692 ;       R5 - Unchanged
                     012B   693 ;
                     012B   694 ; Side Effects:
                     012B   695 ;
                     012B   696 ;       R8 is destroyed
                     012B   697 ;-
                     012B   698
                     012B   699 EO$BLANK_ZERO_ROUTINE:
                     012B   700          MARK_POINT      BLANK_ZERO_1
   58    83    9A    012B   701          MOVZBL  (R3)+,R8                ; Get length
09 5B    02    E1    012E   702          BBC     #PSL$V_Z,R11,20$        ; Skip rest if source string is zero
   55    58    C2    0132   703          SUBL    R8,R5                   ; Back up destination pointer
                     0135   704          MARK_POINT      BLANK_ZERO_2 , RESTART
   85    52    90    0135   705 10$:     MOVB    R2,(R5)+                ; STORE fill character
     FA  58    F5    0138   706          SOBGTR  R8,10$                  ; Check for end of loop
                     013B   707 20$:     RSB
```

```
                        013C   709          .SUBTITLE       EO$REPLACE_SIGN - Replace Sign When Zero
                        013C   710 ;+
                        013C   711 ; Functional Description:
                        013C   712 ;
                        013C   713 ;       If the value of the source string is zero, then the contents of the
                        013C   714 ;       fill register are stored into the byte of the destination string
                        013C   715 ;       that is "length" bytes before the current position.
                        013C   716 ;
                        013C   717 ; Input Parameters:
                        013C   718 ;
                        013C   719 ;       R2 - Fill character
                        013C   720 ;       R3 - Address of "length", number of characters to blank
                        013C   721 ;       R5 - Address of next character to be stored in output character string
                        013C   722 ;       R11<Z> - Set if input string is zero
                        013C   723 ;
                        013C   724 ; Output Parameters:
                        013C   725 ;
                        013C   726 ;       Contents of fill register are stored in byte of output string
                        013C   727 ;       "length" bytes before current position if input string is zero.
                        013C   728 ;
                        013C   729 ;       R3 - Advanced one byte over "length"
                        013C   730 ;       R5 - Unchanged
                        013C   731 ;
                        013C   732 ; Side Effects:
                        013C   733 ;
                        013C   734 ;       R8 is destroyed
                        013C   735 ;-
                        013C   736
                        013C   737 EO$REPLACE_SIGN_ROUTINE:
                        013C   738          MARK_POINT      REPLACE_SIGN_1
     58    83    9A     013C   739          MOVZBL  (R3)+,R8                ; Get length
   07 5B    02    E1     013F   740          BBC     #PSL$V_Z,R11,10$        ; Skip rest if source string is zero
58   55    58    C3     0143   741          SUBL3   R8,R5,R8                ; Get address of indicated byte
                        0147   742          MARK_POINT      REPLACE_SIGN_2
     68    52    90     0147   743          MOVB    R2,(R8)                 ; STORE fill character
                  05    014A   744 10$:     RSB
```

```
                     014B   746          .SUBTITLE       EO$LOAD_xxxxxx - Load Register
                     014B   747 ;+
                     014B   748 ; Functional Description:
                     014B   749 ;
                     014B   750 ;       The contents of the fill or sign register are replaced with the
                     014B   751 ;       character that follows the pattern operator in the pattern stream.
                     014B   752 ;
                     014B   753 ;       EO$LOAD_FILL    Load Fill Register
                     014B   754 ;
                     014B   755 ;       EO$LOAD_SIGN    Load Sign Register
                     014B   756 ;
                     014B   757 ;       EO$LOAD_PLUS    Load Sign Register If Source String Is Positive (or Zero)
                     014B   758 ;
                     014B   759 ;       EO$LOAD_MINUS   Load Sign Register If Source String Is Negative
                     014B   760 ;
                     014B   761 ; Input Parameters:
                     014B   762 ;
                     014B   763 ;       R3 - Address of character to be loaded
                     014B   764 ;       R11<N> - Set if input string is LSS zero (negative)
                     014B   765 ;
                     014B   766 ; Output Parameters:
                     014B   767 ;
                     014B   768 ;       If entry is at EO$LOAD_FILL, the fill register contents (R2<7:0>) are
                     014B   769 ;       replaced with the next character in the pattern stream.
                     014B   770 ;
                     014B   771 ;       If one of the other entry points is used (and the appropriate conditions
                     014B   772 ;       obtain), the contents of the sign register are replaced with the next
                     014B   773 ;       character in the pattern stream. For simplicity of implementation, the
                     014B   774 ;       sign character is stored in R4<7:0> while this routine executes.
                     014B   775 ;
                     014B   776 ;       In the event of an exception, the contents of R4<7:0> will be stored
                     014B   777 ;       in R2<15:8>, either to conform to the architectural specification of
                     014B   778 ;       register contents in the event of a reserved operand fault, or to
                     014B   779 ;       allow the instruction to be restarted in the event of an access
                     014B   780 ;       violation.
                     014B   781 ;
                     014B   782 ;       R3 - Advanced one byte over new fill or sign character
                     014B   783 ;-
                     014B   784
                     014B   785 EO$LOAD_FILL_ROUTINE:
                     014B   786          MARK_POINT      LOAD_xxxx_1
     52    83    90  014B   787          MOVB    (R3)+,R2                ; Load new fill character
                 05  014E   788          RSB
                     014F   789
                     014F   790 EO$LOAD_SIGN_ROUTINE:
                     014F   791          MARK_POINT      LOAD_xxxx_2
     54    83    90  014F   792          MOVB    (R3)+,R4                ; Load new sign character into R4
                 05  0152   793          RSB
                     0153   794
                     0153   795 EO$LOAD_PLUS_ROUTINE:
  F8 5B    03    E1  0153   796          BBC     #PSL$V_N,R11,EO$LOAD_SIGN_ROUTINE ; Use common code if plus
           53    D6  0157   797          INCL    R3                      ; Otherwise, skip unused character
                 05  0159   798          RSB
                     015A   799
                     015A   800 EO$LOAD_MINUS_ROUTINE:
  F1 5B    03    E0  015A   801          BBS     #PSL$V_N,R11,EO$LOAD_SIGN_ROUTINE ; Use common code if minus
           53    D6  015E   802          INCL    R3                      ; Otherwise, skip unused character
```

```
05  0160    803          RSB
```

```
                     0161   805          .SUBTITLE       EO$xxxxxx_SIGNIF - Significance
                     0161   806 ;+
                     0161   807 ; Functional Description:
                     0161   808 ;
                     0161   809 ;       The significance indicator (C-bit in auxiliary PSW) is set or
                     0161   810 ;       cleared according to the entry point.
                     0161   811 ;
                     0161   812 ; Input Parameters:
                     0161   813 ;
                     0161   814 ;       None
                     0161   815 ;
                     0161   816 ; Output Parameters:
                     0161   817 ;
                     0161   818 ;       EO$CLEAR_SIGNIF         R11<C> is cleared
                     0161   819 ;
                     0161   820 ;       EO$SET_SIGNIF           R11<C> is set
                     0161   821 ;-
                     0161   822
                     0161   823 EO$CLEAR_SIGNIF_ROUTINE:
5B   01   8A         0161   824          BICB2   #PSL$M_C,R11            ; Clear significance
          05         0164   825          RSB
                     0165   826
                     0165   827 EO$SET_SIGNIF_ROUTINE:
5B   01   88         0165   828          BISB2   #PSL$M_C,R11            ; Set significance
          05         0168   829          RSB
```

```
                     0169   831          .SUBTITLE       EO_ADJUST_INPUT - Adjust Input Length
                     0169   832 ;+
                     0169   833 ; Functional Description:
                     0169   834 ;
                     0169   835 ;       The pattern operator is followed by an unsigned byte integer length in
                     0169   836 ;       the  range  1  through  31.  If the source string has more digits than
                     0169   837 ;       this length, the excess leading digits are read and discarded.  If any
                     0169   838 ;       discarded  digits  are  non-zero then overflow is set, significance is
                     0169   839 ;       set, and zero is cleared.  If the source string has fewer digits  than
                     0169   840 ;       this  length,  a  counter  is  set  of  the number of leading zeros to
                     0169   841 ;       supply.  This counter is stored as a negative number in R0<31:16>.
                     0169   842 ;-
                     0169   843
                     0169   844 EO$ADJUST_INPUT_ROUTINE:
                     0169   845          MARK_POINT      ADJUST_INPUT_1
     58   83   9A     0169   846          MOVZBL  (R3)+,R8                ; Get "length" from pattern stream
58   50   58   C3     016C   847          SUBL3   R8,R0,R8                ; Is length larger than input length?
          11   1B     0170   848          BLEQU   30$                     ; Branch if yes
          59   D4     0172   849          CLRL    R9                      ; Clear count of zeros ("R0<31:16>")
                      0174   850
                      0174   851 10$:     EO_READ                         ; Get next input digit
          06   13     0177   852          BEQL    20$                     ; Skip to end of loop if zero
     5B   04   8A     0179   853          BICB    #PSL$M_Z,R11            ; Otherwise, indicate nonzero
     5B   03   88     017C   854          BISB    #<PSL$M_C!PSL$M_V>,R11  ; Indicate significance and overflow
       F2 58   F5     017F   855 20$:     SOBGTR  R8,10$                  ; Test for end of loop
          05   0182   856          RSB
                      0183   857
     59   58   D0     0183   858 30$:     MOVL    R8,R9                   ; Store difference into "R0<31:16>"
          05   0186   859          RSB
```

```
0187   861          .SUBTITLE       EO$END - End Edit
0187   862 ;+
0187   863 ; Functional Description:
0187   864 ;
0187   865 ;       The edit operation is terminated.
0187   866 ;
0187   867 ;       The architectural description of EDITPC divides end processing between
0187   868 ;       the EO$END routine and code at the end of the main loop. This
0187   869 ;       implementation performs all of the work in a single place.
0187   870 ;
0187   871 ;       The edit operation is terminated. There are several details that this
0187   872 ;       routine must take care of.
0187   873 ;
0187   874 ;       1.  The return PC to the main dispatch loop is discarded.
0187   875 ;
0187   876 ;       2.  R3 is backed up to point to the EO$END pattern operator.
0187   877 ;
0187   878 ;       3.  A special check must be made for negative zero to insure that
0187   879 ;           the N-bit is cleared.
0187   880 ;
0187   881 ;       4.  If any digits still remain in the input string, a reserved
0187   882 ;           operand abort is taken.
0187   883 ;
0187   884 ;       5.  R2 and R4 are set to zero according to the architecture.
0187   885 ;
0187   886 ; Input Parameters:
0187   887 ;
0187   888 ;       R0 - Number of digits remaining in input string
0187   889 ;       R3 - Address of one byte beyond the EO$END operator
0187   890 ;
0187   891 ;       00(SP)  - Return address in dispatch loop in this module (discarded)
0187   892 ;       04(SP) - Saved R0
0187   893 ;       08(SP) - Saved R1
0187   894 ;       12(SP) - Saved R6
0187   895 ;       16(SP) - Saved R7
0187   896 ;       20(SP) - Saved R8
0187   897 ;       24(SP) - Saved R9
0187   898 ;       28(SP) - Saved R10
0187   899 ;       32(SP) - Saved R11
0187   900 ;       36(SP) - Return PC to caller of VAX$EDITPC
0187   901 ;
0187   902 ; Output Parameters:
0187   903 ;
0187   904 ;       If no overflow has occurred, then this routine exits through the RSB
0187   905 ;       instruction with the following output parameters:
0187   906 ;
0187   907 ;               These register contents are dictated by the VAX architecture
0187   908 ;
0187   909 ;               R0 - Length in digits of input decimal string
0187   910 ;               R1 - Address of most significant byte of input decimal string
0187   911 ;               R2 - Set to zero to conform to architecture
0187   912 ;               R3 - Backed up one byte to point to EO$END operator
0187   913 ;               R4 - Set to zero to conform to architecture
0187   914 ;               R5 - Address of one byte beyond destination character string
0187   915 ;
0187   916 ;               PSL<V> is clear
0187   917 ;
```

```
                     0187   918 ;       If the V-bit is set, then control is transferred to VAX$EDITPC_OVERFLOW
                     0187   919 ;       where a check for decimal overflow exceptions is made.
                     0187   920 ;
                     0187   921 ;       The registers are loaded with their correct contents and then saved on
                     0187   922 ;       the stack as follows:
                     0187   923 ;
                     0187   924 ;               00(SP) - Saved R0
                     0187   925 ;               04(SP) - Saved R1
                     0187   926 ;               08(SP) - Saved R2
                     0187   927 ;               12(SP) - Saved R3
                     0187   928 ;               16(SP) - Saved R4
                     0187   929 ;               20(SP) - Saved R5
                     0187   930 ;               24(SP) - Saved R6
                     0187   931 ;               28(SP) - Saved R7
                     0187   932 ;               32(SP) - Saved R8
                     0187   933 ;               36(SP) - Saved R9
                     0187   934 ;               40(SP) - Saved R10
                     0187   935 ;               44(SP) - Saved R11
                     0187   936 ;               48(SP) - Return PC to caller of VAX$EDITPC
                     0187   937 ;
                     0187   938 ;               PSL<V> is set
                     0187   939 ;-
                     0187   940
                     0187   941 EO$END_ROUTINE:
     5E    04   C0   0187   942         ADDL    #4,SP                   ; Discard return PC to main loop
           53   D7   018A   943         DECL    R3                      ; Back up pattern pointer one byte
  03 5B    02   E1   018C   944         BBC     #PSL$V_Z,R11,10$        ; Check for negative zero
     5B    08   8A   0190   945         BICB    #PSL$M_N,R11            ; Turn off N-bit if zero
           50   D5   0193   946 10$:     TSTL    R0                      ; Any digits remaining?
           5D   12   0195   947         BNEQ    EDITPC_ROPRAND_ABORT    ; Error if yes
           59   D5   0197   948         TSTL    R9                      ; Any zeros (R0<31:16>) remaining?
           59   12   0199   949         BNEQ    EDITPC_ROPRAND_ABORT    ; Error if yes
           52   D4   019B   950         CLRL    R2                      ; Architecture specifies that R2
           54   D4   019D   951         CLRL    R4                      ;  and R4 are zero on exit
           0F   B9   019F   952         BICPSW  #<PSL$M_N!PSL$M_Z!PSL$M_V!PSL$M_C>       ; Clear condition codes
           5B   B8   01A1   953         BISPSW  R11                     ; Set codes according to saved PSW
05 5B      01   E0   01A3   954         BBS     #PSL$V_V,R11,20$        ; Get out of line if overflow
   0FC3    8F   BA   01A7   955         POPR    #^M<R0,R1,R6,R7,R8,R9,R10,R11>  ; Restore saved registers
           05   01AB   956         RSB                                     ; Return to caller's caller
                01AC   957
                01AC   958 ; At this point, we must determine whether the DV bit is set. The tests that
                01AC   959 ; must be performed are identical to the tests performed by the overflow
                01AC   960 ; checking code for the packed decimal routines. In order to make use of
                01AC   961 ; that code, we need to set up the saved registers on the stack to match
                01AC   962 ; the input to that routine. Note also that the decimal routines specify
                01AC   963 ; that R0 is zero on completion while EDITPC dictates that R0 contains the
                01AC   964 ; initial value of "srclen". For this reason, we cannot simply branch to
                01AC   965 ; VAX$DECIMAL_EXIT but must use a special entry point.
                01AC   966
           03   BA   01AC   967 20$:     POPR    #^M<R0,R1>              ; Restore R0 and R1
           3F   BB   01AE   968         PUSHR   #^M<R0,R1,R2,R3,R4,R5>  ; ... only to save them again
                01B0   969
                01B0   970 ; The condition codes were not changed by the previous two instructions.
                01B0   971
         FE4D'  31   01B0   972         BRW     VAX$EDITPC_OVERFLOW     ; Join exit code
```

```
                        .SUBTITLE       EDITPC_ROPRAND_FAULT - Handle Illegal Pattern Operator
                ;+
                ; Functional Description:
                ;
                ;       This routine stores the intermediate state of an EDITPC instruction
                ;       that has been prematurely terminated by an illegal pattern operator.
                ;       These exceptions and access violations are the only exceptions from
                ;       which execution can continue after the exceptional condition has been
                ;       cleared up. After the state is stored in the registers R0 through R5,
                ;       control is transferred through VAX$ROPRAND to VAX$REFLECT_FAULT, where
                ;       the appropriate backup method is determined, based on the return PC
                ;       from the VAX$EDITPC routine.
                ;
                ; Input Parameters:
                ;
                ;       R0  - Current digit count in input string
                ;       R1  - Address of next digit in input string
                ;       R2  - Fill character
                ;       R3  - Address of illegal pattern operator
                ;       R4  - Sign character (stored in R2<15:8>)
                ;       R5  - Address of next character to be stored in output character string
                ;       R9  - Zero count (stored in R0<31:16>)
                ;       R11 - Condition codes
                ;
                ;       00(SP) - Saved R0
                ;       04(SP) - Saved R1
                ;       08(SP) - Saved R6
                ;       12(SP) - Saved R7
                ;       16(SP) - Saved R8
                ;       20(SP) - Saved R9
                ;       24(SP) - Saved R10
                ;       28(SP) - Saved R11
                ;       32(SP) - Return PC from VAX$EDITPC routine
                ;
                ; Output Parameters:
                ;
                ;       00(SP) - Offset in packed register array to delta PC byte
                ;       04(SP) - Return PC from VAX$EDITPC routine
                ;
                ;       Some of the register contents are dictated by the VAX architecture.
                ;       Other register contents are architecturally described as "implementation
                ;       dependent" and are used to store the instruction state that enables it
                ;       to be restarted successfully and complete according to specifications.
                ;
                ;       The following register contents are architecturally specified
                ;
                ;               R0<15:00> - Current digit count in input string
                ;               R0<31:16> - Current zero count (from R9)
                ;               R1        - Address of next digit in input string
                ;               R2<07:00> - Fill character
                ;               R2<15:08> - Sign character (from R4)
                ;               R3        - Address of next pattern operator
                ;               R5        - Address of next character in output character string
                ;
                ;       The following register contents are peculiar to this implementation
                ;
                ;               R2<23:16> - Delta-PC (if initiated by exception)
```

```
                     01B3  1031 ;                  R2<31:24> - Delta srcaddr (current srcaddr minus initial srcaddr)
                     01B3  1032 ;                  R4<07:00> - Initial digit count (from saved R0)
                     01B3  1033 ;                  R4<15:08> - Saved condition codes (for easy retrieval)
                     01B3  1034 ;                  R4<23:16> - State flags
                     01B3  1035 ;                              State = EDITPC_2_RESTART
                     01B3  1036 ;                              FPD bit is set
                     01B3  1037 ;                              ACCVIO bit is clear
                     01B3  1038 ;                  R4<31:24> - Unused for this exception (see access violations)
                     01B3  1039 ;
                     01B3  1040 ;                  EDITPC_2_RESTART is the restart code that causes the instruction
                     01B3  1041 ;                  to be restarted at the top of the main loop. It is the simplest
                     01B3  1042 ;                  point at which to resume execution after an illegal pattern
                     01B3  1043 ;                  operator fault.
                     01B3  1044 ;
                     01B3  1045 ;           The condition codes reported in the exception PSL are also defined
                     01B3  1046 ;           by the VAX architecture.
                     01B3  1047 ;
                     01B3  1048 ;                  PSL<N> - Source string has a minus sign
                     01B3  1049 ;                  PSL<Z> - All digits are zero so far
                     01B3  1050 ;                  PSL<V> - Nonzero digits have been lost
                     01B3  1051 ;                  PSL<C> - Significance
                     01B3  1052 ;-
                     01B3  1053
                     01B3  1054          ASSUME EDITPC_L_SAVED_R1 EQ <EDITPC_L_SAVED_R0 + 4>
                     01B3  1055
                     01B3  1056 EDITPC_ROPRAND_FAULT:
          0F   BB    01B3  1057          PUSHR   #^M<R0,R1,R2,R3>                ; Save current R0..R3
    50 10 AE   7D    01B5  1058          MOVQ    EDITPC_L_SAVED_R0(SP),R0        ; Retrieve original R0 and R1
    10 AE  54  7D    01B9  1059          MOVQ    R4,16(SP)                       ; Save R4 and R5 in right place on s
                     01BD  1060
                     01BD  1061 ; Now start stuffing the various registers
                     01BD  1062
       02 AE  59  B0 01BD  1063          MOVW    R9,EDITPC_W_ZERO_COUNT(SP)      ; Save R9 in R0<31:16>
       09 AE  54  90 01C1  1064          MOVB    R4,EDITPC_B_SIGN(SP)            ; Save R4 in R2<15:8>
       10 AE  50  90 01C5  1065          MOVB    R0,EDITPC_B_INISRCLEN(SP)       ; Save initial value of R0
 51    04 AE  51  C3 01C9  1066          SUBL3   R1,EDITPC_A_SRCADDR(SP),R1      ; Calculate srcaddr difference
       0B AE  51  90 01CE  1067          MOVB    R1,EDITPC_B_DELTA_SRCADDR(SP)   ; Store it in R4<15:8>
       11 AE  5B  90 01D2  1068          MOVB    R11,EDITPC_B_SAVED_PSW(SP)      ; Save condition codes
              12  90 01D6  1069          MOVB    #<EDITPC_M_FPD!EDITPC_2_RESTART>,-
            12 AE        01D8  1070                  EDITPC_B_STATE(SP)          ; Set the FPD bit
                     01DA  1071
          0FFF 8F  BA 01DA  1072          POPR    #^M<R0,R1,R2,R3,R4,R5,R6,R7,R8,R9,R10,R11> ; Load registers
      0000010A 8F  DD 01DE  1073          PUSHL   #<EDITPC_B_DELTA_PC!-           ; Store delta-PC offset
                     01E4  1074                  PACK_M_FPD>                     ; Indicate that FPD should be set
                     01E4  1075
                     01E4  1076 ; The following is admittedly gross. This is the only code path into
                     01E4  1077 ; VAX$ROPRAND where the condition codes are significant. All other paths can
                     01E4  1078 ; store the delta-PC offset without concern for its affect on condition
                     01E4  1079 ; codes. Fortunately, the POPR instruction does not affect condition codes.
                     01E4  1080
                     01E4  1081          ASSUME EDITPC_B_SAVED_PSW EQ 17 ; Make sure we get them from right place
                     01E4  1082
              50  DD 01E4  1083          PUSHL   R0                              ; Get a scratch register
 50   54  04  08  EF 01E6  1084          EXTZV   #8,#4,R4,R0                     ; Get codes from R4<11:8>
              0F  B9 01EB  1085          BICPSW  #<PSL$M_N!PSL$M_Z!PSL$M_V!PSL$M_C>      ; Clear the codes
              50  B8 01ED  1086          BISPSW  R0                              ; Set relevant condition codes
              01  BA 01EF  1087          POPR    #^M<R0>                         ; Restore R0, preserving PSW
```

```
FE0C'  31  01F1  1088          BRW     VAX$ROPRAND             ; Continue exception handling
```

```
01F4  1090          .SUBTITLE       EDITPC_ROPRAND_ABORT - Abnormally Terminate Instruction
01F4  1091 ;-
01F4  1092 ; Functional Description:
01F4  1093 ;
01F4  1094 ;       This routine reports a reserved operand abort back to the caller.
01F4  1095 ;
01F4  1096 ;       Reserved operand aborts are trivial to handle because they cannot be
01F4  1097 ;       continued. There is no need to pack intermediate state into the
01F4  1098 ;       general registers. Those registers that should not be modified by the
01F4  1099 ;       EDITPC instruction have their contents restored. Control is then
01F4  1100 ;       passed to VAX$ROPRAND, which takes the necessary steps to eventually
01F4  1101 ;       reflect the exception back to the caller.
01F4  1102 ;
01F4  1103 ;       The following conditions cause a reserved operand abort
01F4  1104 ;
01F4  1105 ;           1.  Input digit count GTRU 31
01F4  1106 ;               (This condition is detected by the EDITPC initialization code.)
01F4  1107 ;
01F4  1108 ;           2.  Not enough digits in source string to satisfy pattern operators
01F4  1109 ;               (This condition is detected by the EO_READ routine.)
01F4  1110 ;
01F4  1111 ;           3.  Too many digits in source string (digits left over)
01F4  1112 ;               (This condition is detected by the EO$END routine.)
01F4  1113 ;
01F4  1114 ;           4.  An EO$END operator was encountered while zero count was nonzero
01F4  1115 ;               (This condition is also detected by the EO$END routine.)
01F4  1116 ;
01F4  1117 ; Input Parameters:
01F4  1118 ;
01F4  1119 ;       00(SP) - Saved R0
01F4  1120 ;       04(SP) - Saved R1
01F4  1121 ;       08(SP) - Saved R6
01F4  1122 ;       12(SP) - Saved R7
01F4  1123 ;       16(SP) - Saved R8
01F4  1124 ;       20(SP) - Saved R9
01F4  1125 ;       24(SP) - Saved R10
01F4  1126 ;       28(SP) - Saved P11
01F4  1127 ;       32(SP) - Return PC from VAX$EDITPC routine
01F4  1128 ;
01F4  1129 ; Output Parameters:
01F4  1130 ;
01F4  1131 ;       The contents of R0 through R5 are not important because the
01F4  1132 ;       architecture states that they are UNPREDICTABLE if a reserved
01F4  1133 ;       operand abort occurs. No effort is made to put these registers
01F4  1134 ;       into a consistent state.
01F4  1135 ;
01F4  1136 ;       R6 through R11 are restored to their values when the EDITPC
01F4  1137 ;       instruction began executing.
01F4  1138 ;
01F4  1139 ;       00(SP) - Offset in packed register array to delta PC byte
01F4  1140 ;       04(SP) - Return PC from VAX$EDITPC routine
01F4  1141 ;
01F4  1142 ; Implicit Output:
01F4  1143 ;
01F4  1144 ;       This routine passes control to VAX$ROPRAND where further
01F4  1145 ;       exception processing takes place.
01F4  1146 ;-
```

```
                     01F4  1147
                     01F4  1148 EDITPC_ROPRAND_ABORT:
0FC3 8F    BA        01F4  1149          POPR    #^M<R0,R1,R6,R7,R8,R9,R10,R11>  ; Restore saved registers
       0A  DD        01F8  1150          PUSHL   #EDITPC_B_DELTA_PC              ; Store delta-PC offset
   FE03'   31        01FA  1151          BRW     VAX$ROPRAND                     ; Continue exception handling
```

```
01FD  1153          .SUBTITLE       EDITPC_ACCVIO - Reflect an Access Violation
01FD  1154 ;+
01FD  1155 ; Functional Description:
01FD  1156 ;
01FD  1157 ;       This routine receives control when an access violation occurs while
01FD  1158 ;       executing within the EDITPC emulator. This routine determines whether
01FD  1159 ;       the exception occurred while accessing the source decimal string, the
01FD  1160 ;       pattern stream, or the output character string. (This check is made
01FD  1161 ;       based on the PC of the exception.)
01FD  1162 ;
01FD  1163 ;       If the PC is one that is recognized by this routine, then the state of
01FD  1164 ;       the instruction (character counts, string addresses, and the like) are
01FD  1165 ;       restored to a state where the instruction/routine can be restarted
01FD  1166 ;       after (if) the cause for the exception is eliminated. Control is then
01FD  1167 ;       passed to a common routine that sets up the stack and the exception
01FD  1168 ;       parameters in such a way that the instruction or routine can restart
01FD  1169 ;       transparently.
01FD  1170 ;
01FD  1171 ;       If the exception occurs at some unrecognized PC, then the exception is
01FD  1172 ;       reflected to the user as an exception that occurred within the
01FD  1173 ;       emulator.
01FD  1174 ;
01FD  1175 ;       There are two exceptions that can occur that are not backed up to
01FD  1176 ;       appear as if they occurred at the site of the original emulated
01FD  1177 ;       instruction. These exceptions will appear to the user as if they
01FD  1178 ;       occurred inside the emulator itself.
01FD  1179 ;
01FD  1180 ;           1.  If stack overflow occurs due to use of the stack by one of
01FD  1181 ;               the routines, it is unlikely that this routine will even
01FD  1182 ;               execute because the code that transfers control here must
01FD  1183 ;               first copy the parameters to the exception stack and that
01FD  1184 ;               operation would fail. (The failure causes control to be
01FD  1185 ;               transferred to VMS, where the stack expansion logic is
01FD  1186 ;               invoked and the routine resumed transparently.)
01FD  1187 ;
01FD  1188 ;           2.  If assumptions about the address space change out from under
01FD  1189 ;               these routines (because an AST deleted a portion of the
01FD  1190 ;               address space or a similar silly thing), the handling of the
01FD  1191 ;               exception is UNPREDICTABLE.
01FD  1192 ;
01FD  1193 ; Input Parameters:
01FD  1194 ;
01FD  1195 ;       R0  - Value of SP when exception occurred
01FD  1196 ;       R1  - PC at which exception occurred
01FD  1197 ;       R2  - scratch
01FD  1198 ;       R3  - scratch
01FD  1199 ;       R10 - Address of this routine (no longer needed)
01FD  1200 ;
01FD  1201 ;       00(SP) - Value of R0 when exception occurred
01FD  1202 ;       04(SP) - Value of R1 when exception occurred
01FD  1203 ;       08(SP) - Value of R2 when exception occurred
01FD  1204 ;       12(SP) - Value of R3 when exception occurred
01FD  1205 ;       16(SP) - Return PC in exception dispatcher in operating system
01FD  1206 ;
01FD  1207 ;       20(SP) - First longword of system-specific exception data
01FD  1208 ;         .
01FD  1209 ;         .
```

```
01FD  1210 ;       xx(SP) - Last longword of system-specific exception data
01FD  1211 ;
01FD  1212 ;       The address of the next longword is the position of the stack when
01FD  1213 ;       the exception occurred. R0 locates this address.
01FD  1214 ;
01FD  1215 ; R0 -> xx+4(SP)     - Instruction-specific data
01FD  1216 ;         .          - Optional instruction-specific data
01FD  1217 ;         .          - Optional instruction-specific data
01FD  1218 ;       xx+<4*M>(SP) - Return PC from VAX$EDITPC routine (M is the number
01FD  1219 ;                      of instruction-specific longwords)
01FD  1220 ;
01FD  1221 ; Implicit Input:
01FD  1222 ;
01FD  1223 ;       It is assumed that the contents of all registers coming into this
01FD  1224 ;       routine are unchanged from their contents when the exception occurred.
01FD  1225 ;       (For R0 through R3, this assumption applies to the saved register
01FD  1226 ;       contents on the top of the stack. Any modification to these four
01FD  1227 ;       registers must be made to their saved copies and not to the registers
01FD  1228 ;       themselves.)
01FD  1229 ;
01FD  1230 ;       It is further assumed that the exception PC is within the bounds of
01FD  1231 ;       this module. (Violation of this assumption is simply an inefficiency.)
01FD  1232 ;
01FD  1233 ;       Finally, the macro BEGIN_MARK_POINT should have been invoked at the
01FD  1234 ;       beginning of this module to define the symbols
01FD  1235 ;
01FD  1236 ;               MODULE_BASE
01FD  1237 ;               PC_TABLE_BASE
01FD  1238 ;               HANDLER_TABLE_BASE
01FD  1239 ;               TABLE_SIZE
01FD  1240 ;
01FD  1241 ; Output Parameters:
01FD  1242 ;
01FD  1243 ;       If the exception is recognized (that is, if the exception PC is
01FD  1244 ;       associated with one of the mark points), control is passed to the
01FD  1245 ;       context-specific routine that restores the instruction state to a
01FD  1246 ;       uniform point from which the EDITPC instruction can be restarted.
01FD  1247 ;
01FD  1248 ;               R0  - Value of SP when exception occurred
01FD  1249 ;               R1  - scratch
01FD  1250 ;               R2  - scratch
01FD  1251 ;               R3  - scratch
01FD  1252 ;               R10 - scratch
01FD  1253 ;
01FD  1254 ;       VAX$EDITPC is different from the other emulated instructions in that
01FD  1255 ;       it requires intermediate state to be stored in R4 and R5 as well as R0
01FD  1256 ;       through R3. This requires that R4 and R5 also be saved on the stack so
01FD  1257 ;       that they can be manipulated in a consistent fashion.
01FD  1258 ;
01FD  1259 ;       00(SP) - Value of R0 when exception occurred
01FD  1260 ;       04(SP) - Value of R1 when exception occurred
01FD  1261 ;       08(SP) - Value of R2 when exception occurred
01FD  1262 ;       12(SP) - Value of R3 when exception occurred
01FD  1263 ;       16(SP) - Value of R4 when exception occurred
01FD  1264 ;       20(SP) - Value of R5 when exception occurred
01FD  1265 ;       24(SP) - Value of R0 when exception occurred
01FD  1266 ;       28(SP) - Value of R1 when exception occurred
```

```
                                01FD  1267 ;       32(SP) - Value of R2 when exception occurred
                                01FD  1268 ;       36(SP) - Value of R3 when exception occurred
                                01FD  1269 ;       40(SP) - Return PC in exception dispatcher in operating system
                                01FD  1270 ;        etc.
                                01FD  1271 ;
                                01FD  1272 ; R0 -> zz(SP) - Instruction-specific data begins here
                                01FD  1273 ;
                                01FD  1274 ;       If the exception PC occurred somewhere else (such as a stack access),
                                01FD  1275 ;       the saved registers are restored and control is passed back to the
                                01FD  1276 ;       host system with an RSB instruction.
                                01FD  1277 ;
                                01FD  1278 ; Implicit Output:
                                01FD  1279 ;
                                01FD  1280 ;       The register contents are modified to put the intermediate state of
                                01FD  1281 ;       the instruction into a consistent state from which it can be
                                01FD  1282 ;       continued. Any registers saved by the VAX$EDITPC routine are
                                01FD  1283 ;       restored.
                                01FD  1284 ;-
                                01FD  1285
                                01FD  1286         .ENABLE         LOCAL_BLOCK
                                01FD  1287
                                01FD  1288 EDITPC_ACCVIO:
          7E    54    7D      01FD  1289         MOVQ    R4,-(SP)                ; Store R5 and R4 on the stack
       7E   10 AE    7D      0200  1290         MOVQ    16(SP),-(SP)            ; ... and another copy of R3 and R2
       7E   10 AE    7D      0204  1291         MOVQ    16(SP),-(SP)            ; ... and another copy of R1 and R0
                             0208  1292
             52    D4        0208  1293         CLRL    R2                      ; Initialize the counter
          FDF2 CF    9F      020A  1294         PUSHAB  MODULE_BASE             ; Store base address of this module
          51    8E    C2    020E  1295         SUBL2   (SP)+,R1                ; Get PC relative to this base
                             0211  1296
  0000'CF42  51    B1        0211  1297 10$:    CMPW    R1,PC_TABLE_BASE[R2]    ; Is this the right PC?
             0D    13        0217  1298         BEQL    30$                     ; Exit loop if true
       F4 52    1B    F2    0219  1299         AOBLSS  #TABLE_SIZE,R2,10$      ; Do the entire table
                             021D  1300
                             021D  1301 ; If we drop through the dispatching based on PC, then the exception is not
                             021D  1302 ; one that we want to back up. We simply reflect the exception to the user.
                             021D  1303
          5E    10    C0      021D  1304 20$:    ADDL    #16,SP                  ; Discard duplicate saved R0 .. R3
          54    8E    7D      0220  1305         MOVQ    (SP)+,R4                ; Restore R4 and R5
                0F    BA      0223  1306         POPR    #^M<R0,R1,R2,R3>        ; Restore saved registers
                05    05      0225  1307         RSB                             ; Return to exception dispatcher
                             0226  1308
                             0226  1309 ; The exception PC matched one of the entries in our PC table. R2 contains
                             0226  1310 ; the index into both the PC table and the handler table. R1 has served
                             0226  1311 ; its purpose and can be used as a scratch register.
                             0226  1312
  51  0000'CF42    3C        0226  1313 30$:    MOVZWL  HANDLER_TABLE_BASE[R2],R1       ; Get the offset to the handler
      FDCF CF41    17        022C  1314         JMP     MODULE_BASE[R1]         ; Pass control to the handler
                             0231  1315
                             0231  1316 ; In all of the instruction-specific routines, the state of the stack
                             0231  1317 ; will be shown as it was when the exception occurred. All offsets will
                             0231  1318 ; be pictured relative to R0.
```

```
                     0231  1320          .SUBTITLE       Access Violation While Reading Input Digit
                     0231  1321 ;+
                     0231  1322 ; EO_READ Packing Routine
                     0231  1323 ;
                     0231  1324 ; Functional Description:
                     0231  1325 ;
                     0231  1326 ;       This routine executes if an access violation occurred in the EO_READ
                     0231  1327 ;       subroutine while accessing the input packed decimal string.
                     0231  1328 ;
                     0231  1329 ; Input Parameters:
                     0231  1330 ;
                     0231  1331 ;       R0 - Address of top of stack when access violation occurred
                     0231  1332 ;
                     0231  1333 ;       00(R0) - Return PC to caller of EO_READ
                     0231  1334 ;       04(R0) - Return PC to main VAX$EDITPC control loop
                     0231  1335 ;       08(R0) - Saved R0
                     0231  1336 ;       12(R0) - Saved R1
                     0231  1337 ;         etc.
                     0231  1338 ;
                     0231  1339 ; Output Parameters:
                     0231  1340 ;
                     0231  1341 ;       If the caller of this routine a recognized restart point, the restart
                     0231  1342 ;       code is stored in EDITPC_B_STATE in the saved register array, the
                     0231  1343 ;       psuedo stack pointer R0 is advanced by one, and control is passed to
                     0231  1344 ;       the general EDITPC_PACK routine for final exception processing.
                     0231  1345 ;
                     0231  1346 ;               R0 is advanced by one longword
                     0231  1347 ;
                     0231  1348 ;               00(R0) - Return PC to main VAX$EDITPC control loop
                     0231  1349 ;               04(R0) - Saved R0
                     0231  1350 ;               08(R0) - Saved R1
                     0231  1351 ;                 etc.
                     0231  1352 ;
                     0231  1353 ;               EDITPC_B_STATE(SP) - Code that uniquely determines the caller
                     0231  1354 ;                       of EO_READ when the access violation was detected.
                     0231  1355 ;
                     0231  1356 ;       If the caller's PC is not recognized, the exception is dismissed from
                     0231  1357 ;       further modification.
                     0231  1358 ;-
                     0231  1359
                     0231  1360 READ_1:
                     0231  1361 READ_2:
                 52  D4  0231  1362          CLRL    R2                              ; Set table index to zero
           FDC9  CF  9F  0233  1363          PUSHAB  MODULE_BASE                     ; Prepare for PIC arithmetic
        51  80  8E  C3  0237  1364          SUBL3   (SP)+,(R0)+,R1                  ; R1 contains relative PC
            51  03  C2  023B  1365          SUBL2   #3,R1                           ; Back up over BSBW instruction
                        023E  1366
0000'CF42   51  B1  023E  1367 40$:       CMPW    R1,RESTART_PC_TABLE_BASE[R2]    ; Check next PC offset
                06  13  0244  1368          BEQL    50$                             ; Exit loop if match
          F4 52 0C  F2  0246  1369          AOBLSS  #RESTART_TABLE_SIZE,R2,40$      ; Check for end of loop
                        024A  1370
                        024A  1371 ; If we drop through this loop, we got into the EO_READ subroutine from
                        024A  1372 ; other than one of the three known call sites. We pass control back to
                        024A  1373 ; the general exception dispatcher.
                        024A  1374
                D1  11  024A  1375          BRB     20$                             ; Join common code to dismiss exception
                        024C  1376
```

```
                     024C  1377 ; Store the restart code appropriate to the return PC and join common code to
                     024C  1378 ; store the rest of the instruction state into the saved register array.
                     024C  1379
                     024C  1380         ASSUME EDITPC_V_STATE EQ 0
                     024C  1381
12 AE   52   01   81 024C  1382 50$:    ADDB3   #1,R2,EDITPC_B_STATE(SP)        ; Restart code base is 1, not 0
                6E   B6 0251  1383         INCW    EDITPC_W_SRCLEN(SP)         ; Digit never got read
                2E   11 0253  1384         BRB     70$                         ; Make sure that R8 is saved
```

```
                     0255  1386          .SUBTITLE       Access Violation While Executing Loop
                     0255  1387 ;+
                     0255  1388 ; Packing Routine for Storage Loops
                     0255  1389 ;
                     0255  1390 ; Functional Description:
                     0255  1391 ;
                     0255  1392 ;       All of the following labels are associated with exceptions that occur
                     0255  1393 ;       inside a loop that is reading digits from the input stream and
                     0255  1394 ;       optionally storing these or other characters in the output string.
                     0255  1395 ;       While it is a trivial matter to back up the output pointer and restart
                     0255  1396 ;       the loop from the beginning, it is somewhat more difficult to handle
                     0255  1397 ;       all of the cases that can occur with the input packed decimal string
                     0255  1398 ;       (because a byte can contain two digits). To avoid this complication,
                     0255  1399 ;       we add the ability to restart the various loops where they left off.
                     0255  1400 ;       In order to accomplish this, we need to store the loop count and,
                     0255  1401 ;       optionally, the latest input digit in the intermediate state array.
                     0255  1402 ;
                     0255  1403 ;       The two entry points where the contents of R7 (the last digit read
                     0255  1404 ;       from the input stream) are significant are MOVE_2 and FLOAT_3. All
                     0255  1405 ;       other entry points ignore the contents of R7. (Note that these two
                     0255  1406 ;       entry points exit through label 60$ to store R7 in the saved register
                     0255  1407 ;       array.)
                     0255  1408 ;
                     0255  1409 ; Input Parameters:
                     0255  1410 ;
                     0255  1411 ;       R0 - Address of top of stack when access violation occurred
                     0255  1412 ;       R7 - Latest digit read from input stream (MOVE_2 and FLOAT_3 only)
                     0255  1413 ;       R8 - Remaining loop count
                     0255  1414 ;
                     0255  1415 ;       00(R0) - Return PC to main VAX$EDITPC control loop
                     0255  1416 ;       04(R0) - Saved R0
                     0255  1417 ;       08(R0) - Saved R1
                     0255  1418 ;         etc.
                     0255  1419 ;
                     0255  1420 ; Output Parameters:
                     0255  1421 ;
                     0255  1422 ;       A restart code that is unique for each entry is stored in the saved
                     0255  1423 ;       register array. The loop count (and the latest input digit, if
                     0255  1424 ;       appropriate) is also stored before passing control to EDITPC_PACK.
                     0255  1425 ;
                     0255  1426 ;       EDITPC_B_STATE(SP) - Code that uniquely determines the code that
                     0255  1427 ;               was executing when the access violation was detected.
                     0255  1428 ;
                     0255  1429 ;       EDITPC_B_EO_READ_CHAR(SP) - Latest digit read from the input string
                     0255  1430 ;
                     0255  1431 ;       EDITPC_B_LOOP_COUNT(SP) - Remaining loop count
                     0255  1432 ;
                     0255  1433 ; Side Effects:
                     0255  1434 ;
                     0255  1435 ;       R0 is unchanged by this code path
                     0255  1436 ;-
                     0255  1437
                     0255  1438          ASSUME EDITPC_V_STATE EQ 0
                     0255  1439
                     0255  1440 FILL_2:
12 AE    03    90    0255  1441          MOVB    #FILL_2_RESTART,EDITPC_B_STATE(SP)
         28    11    0259  1442          BRB     70$
```

```
                                025B  1443
                                025B  1444 MOVE_2:
12 AE   05   90                 025B  1445          MOVB    #MOVE_2_RESTART,EDITPC_B_STATE(SP)
        1E   11                 025F  1446          BRB     60$
                                0261  1447
                                0261  1448 MOVE_3:
12 AE   06   90                 0261  1449          MOVB    #MOVE_3_RESTART,EDITPC_B_STATE(SP)
        1C   11                 0265  1450          BRB     70$
                                0267  1451
                                0267  1452 FLOAT_2:
12 AE   08   90                 0267  1453          MOVB    #FLOAT_2_RESTART,EDITPC_B_STATE(SP)
        12   11                 026B  1454          BRB     60$
                                026D  1455
                                026D  1456 FLOAT_3:
12 AE   09   90                 026D  1457          MOVB    #FLOAT_3_RESTART,EDITPC_B_STATE(SP)
        0C   11                 0271  1458          BRB     60$
                                0273  1459
                                0273  1460 FLOAT_4:
12 AE   0A   90                 0273  1461          MOVB    #FLOAT_4_RESTART,EDITPC_B_STATE(SP)
        0A   11                 0277  1462          BRB     70$
                                0279  1463
                                0279  1464 BLANK_ZERO_2:
12 AE   0B   90                 0279  1465          MOVB    #BLANK_ZERO_2_RESTART,EDITPC_B_STATE(SP)
        04   11                 027D  1466          BRB     70$
                                027F  1467
01 AE   57   90                 027F  1468 60$:     MOVB    R7,EDITPC_B_EO_READ_CHAR(SP)    ; Save result of latest read
13 AE   58   90                 0283  1469 70$:     MOVB    R8,EDITPC_B_LOOP_COUNT(SP)      ; Save loop counter
        16   11                 0287  1470          BRB     80$
```

```
                     0289  1472          .SUBTITLE       Access Violation in Initialization Code
                     0289  1473 ;+
                     0289  1474 ; Functional Description:
                     0289  1475 ;
                     0289  1476 ;       An access violation at EDITPC_1 indicates that the byte containing the
                     0289  1477 ;       sign of the input packed decimal string could not be read. There is
                     0289  1478 ;       little state to preserve. The key step here is to store a restart code
                     0289  1479 ;       that differentiates this exception from the large number that can be
                     0289  1480 ;       restarted at the top of the command loop.
                     0289  1481 ;
                     0289  1482 ; Input Parameters:
                     0289  1483 ;
                     0289  1484 ;       00(R0) - Saved R0
                     0289  1485 ;       04(R0) - Saved R1
                     0289  1486 ;         etc.
                     0289  1487 ;
                     0289  1488 ; Output Parameter:
                     0289  1489 ;
                     0289  1490 ;       EDITPC_B_STATE(SP) - Code that indicates that instruction should
                     0289  1491 ;                            be restarted at point where sign "digit" is fetched.
                     0289  1492 ;-
                     0289  1493          ASSUME EDITPC_V_STATE EQ 0
                     0289  1494
                     0289  1495 EDITPC_1:
12 AE    01    90    0289  1496          MOVB    #EDITPC_1_RESTART,EDITPC_B_STATE(SP)
         13    11    028D  1497          BRB     EDITPC_PACK
```

```
                        028F  1499          .SUBTITLE       Simple Access Violation
                        028F  1500 ;+
                        028F  1501 ; Functional Description:
                        028F  1502 ;
                        028F  1503 ;       This routine handles all of the simple access violations, those that
                        028F  1504 ;       can be backed up to the same intermediate state. In general, an access
                        028F  1505 ;       violation occurred in one of the simpler routines or at some other
                        028F  1506 ;       point where it is not difficult to back up the EDITPC operation to the
                        028F  1507 ;       top of the main dispatch loop.
                        028F  1508 ;
                        028F  1509 ; Input Parameters:
                        028F  1510 ;
                        028F  1511 ;       R3 - Points one byte beyond current pattern operator (except for
                        028F  1512 ;            REPLACE_SIGN_2 where it is one byte further along)
                        028F  1513 ;
                        028F  1514 ;       00(R0) - TOP_OF_LOOP (Return PC to main VAX$EDITPC control loop)
                        028F  1515 ;       04(R0) - Saved R0
                        028F  1516 ;       08(R0) - Saved R1
                        028F  1517 ;         etc.
                        028F  1518 ;
                        028F  1519 ; Output Parameters:
                        028F  1520 ;
                        028F  1521 ;       R3 must be decremented to point to the pattern operator that was being
                        028F  1522 ;       processed when the exception occurred. The return PC must be
                        028F  1523 ;       "discarded" to allow the registers to be restored and the return PC
                        028F  1524 ;       from VAX$EDITPC to be located.
                        028F  1525 ;
                        028F  1526 ;              R3 - Points to current pattern operator
                        028F  1527 ;
                        028F  1528 ;              00(R0) - Saved R0
                        028F  1529 ;              04(R0) - Saved R1
                        028F  1530 ;                etc.
                        028F  1531 ;
                        028F  1532 ; Output Parameter:
                        028F  1533 ;
                        028F  1534 ;       EDITPC_B_STATE(SP) - The restart point called EDITPC_2 is the place
                        028F  1535 ;              from which all "simple" access violations are restarted.
                        028F  1536 ;              This is essentially the location TOP_OF_LOOP.
                        028F  1537 ;-
                        028F  1538
                        028F  1539 END_FLOAT_1:
05 5B   00    E4        028F  1540          BBSC    #PSL$V_C,R11,75$        ; Clear saved C-bit before restarting
        03    11        0293  1541          BRB     75$                     ; We should never get here but ...
                        0295  1542
                        0295  1543 REPLACE_SIGN_2:
     0C AE    D7        0295  1544          DECL    EDITPC_A_PATTERN(SP)    ; Back up to "length" byte
                        0298  1545
                        0298  1546 EDITPC_3:
                        0298  1547 EDITPC_4:
                        0298  1548 EDITPC_5:
                        0298  1549
                        0298  1550 INSERT_1:
                        0298  1551 INSERT_2:
                        0298  1552
                        0298  1553 STORE_SIGN_1:
                        0298  1554
                        0298  1555 FILL_1:
```

```
                     0298  1556
                     0298  1557 MOVE_1:
                     0298  1558
                     0298  1559 FLOAT_1:
                     0298  1560
                     0298  1561 BLANK_ZERO_1:
                     0298  1562
                     0298  1563 REPLACE_SIGN_1:
                     0298  1564
                     0298  1565 LOAD_xxxx_1:
                     0298  1566 LOAD_xxxx_2:
                     0298  1567
                     0298  1568 ADJUST_INPUT_1:
                     0298  1569
  0C AE    D7        0298  1570 75$:     DECL    EDITPC_A_PATTERN(SP)    ; Back up to current pattern operator
                     029B  1571
                     029B  1572 EDITPC_2:
     02    90        029B  1573          MOVB    #EDITPC_2_RESTART,-     ; Store special restart code
  12 AE               029D  1574                  EDITPC_B_STATE(SP)
50   04    C0        029F  1575 80$:     ADDL    #4,R0                   ; Discard return PC
                     02A2  1576                                          ; ... and drop through to EDITPC_PACK
                     02A2  1577
                     02A2  1578          .DISABLE        LOCAL_BLOCK
```

```
                        .SUBTITLE       EDITPC_PACK - Store EDITPC Intermediate State
                ;+
                ; Functional Description:
                ;
                ;       This routine stores the intermediate state of an EDITPC instruction
                ;       that has been prematurely terminated by an access violation. These
                ;       exceptions and illegal pattern operators are the only exceptions from
                ;       which execution can continue after the exceptional condition has been
                ;       cleared up. After the state is stored in the registers R0 through R5,
                ;       control is transferred to VAX$REFLECT_FAULT, where the appropriate
                ;       backup method is determined, based on the return PC from the
                ;       VAX$EDITPC routine.
                ;
                ; Input Parameters:
                ;
                ;       R0  - Current digit count in input string
                ;       R1  - Address of next digit in input string
                ;       R2  - Fill character
                ;       R3  - Address of current pattern operator
                ;       R4  - Sign character (stored in R2<15:8>)
                ;       R5  - Address of next character to be stored in output character string
                ;       R9  - Zero count (stored in R0<31:16>)
                ;       R11 - Condition codes
                ;
                ;       00(R0) - Saved R0
                ;       04(R0) - Saved R1
                ;       08(R0) - Saved R6
                ;       12(R0) - Saved R7
                ;       16(R0) - Saved R8
                ;       20(R0) - Saved R9
                ;       24(R0) - Saved R10
                ;       28(R0) - Saved R11
                ;       32(R0) - Return PC from VAX$EDITPC routine
                ;
                ; Output Parameters:
                ;
                ;       R0 - Address of return PC from VAX$EDITPC routine
                ;
                ;       00(R0) - Return PC from VAX$EDITPC routine
                ;
                ;       Some of the register contents are dictated by the VAX architecture.
                ;       Other register contents are architecturally described as "implementation
                ;       dependent" and are used to store the instruction state that enables it
                ;       to be restarted successfully and complete according to specifications.
                ;
                ;       The following register contents are architecturally specified
                ;
                ;               R0<15:00> - Current digit count in input string
                ;               R0<31:16> - Current zero count (from R9)
                ;               R1        - Address of next digit in input string
                ;               R2<07:00> - Fill character
                ;               R2<15:08> - Sign character (from R4)
                ;               R3        - Address of current pattern operator
                ;               R5        - Address of next character in output character string
                ;
                ;       The following register contents are peculiar to this implementation
                ;
```

```
                        02A2  1637 ;               R2<23:16> - Delta-PC (if initiated by exception)
                        02A2  1638 ;               R2<31:24> - Delta src ddr (current srcaddr minus initial srcaddr)
                        02A2  1639 ;               R4<07:00> - Initial digit count (from saved R0)
                        02A2  1640 ;               R4<15:08> - Saved condition codes (for easy retrieval)
                        02A2  1641 ;               R4<23:16> - State flags
                        02A2  1642 ;                               State field determines the restart point
                        02A2  1643 ;                               FPD bit is set
                        02A2  1644 ;                               ACCVIO bit is set
                        02A2  1645 ;               R4<31:24> - Unused for this exception (see access violations)
                        02A2  1646 ;
                        02A2  1647 ;       The condition codes are not architecturally specified by the VAX
                        02A2  1648 ;       architecture for an access violation. The following list applies to
                        02A2  1649 ;       some but not all of the points where an access violation can occur.
                        02A2  1650 ;
                        02A2  1651 ;               PSL<N> - Source string has a minus sign
                        02A2  1652 ;               PSL<Z> - All digits are zero so far
                        02A2  1653 ;               PSL<V> - Nonzero digits have been lost
                        02A2  1654 ;               PSL<C> - Significance
                        02A2  1655 ;-
                        02A2  1656
                        02A2  1657         ASSUME EDITPC_L_SAVED_R1 EQ <EDITPC_L_SAVED_R0 + 4>
                        02A2  1658
                        02A2  1659 EDITPC_PACK:
                        02A2  1660
                        02A2  1661 ; Now start stuffing the various registers
                        02A2  1662
      02 AE    59   B0  02A2  1663         MOVW    R9,EDITPC_W_ZERO_COUNT(SP)      ; Save R9 in R0<31:16>
      09 AE    54   90  02A6  1664         MOVB    R4,EDITPC_B_SIGN(SP)            ; Save R4 in R2<15:8>
         52    80   7D  02AA  1665         MOVQ    (R0)+,R2                        ; Get initial R0/R1 to R2/R3
      10 AE    52   90  02AD  1666         MOVB    R2,EDITPC_B_INISRCLEN(SP)       ; Save initial value of R0
53    04 AE    53   C3  02B1  1667         SUBL3   R3,EDITPC_A_SRCADDR(SP),R3      ; Calculate srcaddr difference
      0B AE    53   90  02B6  1668         MOVB    R3,EDITPC_B_DELTA_SRCADDR(SP)   ; Store it in R4<15:8>
      11 AE    5B   90  02BA  1669         MOVB    R11,EDITPC_B_SAVED_PSW(SP)      ; Save condition codes
      12 AE    10   88  02BE  1670         BISB    #EDITPC_M_FPD,EDITPC_B_STATE(SP)        ; Set the FPD bit
                        02C2  1671
                        02C2  1672 ; Restore the remaining registers
                        02C2  1673
         56    80   7D  02C2  1674         MOVQ    (R0)+,R6                        ; Restore R6 and R7
         58    80   7D  02C5  1675         MOVQ    (R0)+,R8                        ; ... and R8 and R9
         5A    80   7D  02C8  1676         MOVQ    (R0)+,R10                       ; ... and R10 and R11
                        02CB  1677
                        02CB  1678 ; Get rid of the extra copy of saved registers on the stack
                        02CB  1679
      10 AE    8E   7D  02CB  1680         MOVQ    (SP)+,16(SP)            ; Copy the saved R0/R1 pair
      10 AE    8E   7D  02CF  1681         MOVQ    (SP)+,16(SP)            ; ... and the saved R2/R3 pair
         54    8E   7D  02D3  1682         MOVQ    (SP)+,R4                ; R4 and R5 can be themselves
                        02D6  1683
                        02D6  1684 ; R1 contains delta-PC offset and indicates that FPD gets set
                        02D6  1685
51    0000030A 8F   D0  02D6  1686         MOVL    #<EDITPC_B_DELTA_PC!-   ; Locate delta-PC offset
                        02DD  1687                 PACK_M_FPD!-            ; Set FPD bit in exception PSL
                        02DD  1688                 PACK_M_ACCVIO>,R1       ; Indicate an access violation
            FD20'  31   02DD  1689         BRW     VAX$REFLECT_FAULT       ; Reflect fault to caller
```

```
02E0  1691          .SUBTITLE       EDITPC_RESTART - Unpack and Restart EDITPC Instruction
02E0  1692 ;+
02E0  1693 ; Functional Description:
02E0  1694 ;
02E0  1695 ;       This routine receives control when an EDITPC instruction is restarted.
02E0  1696 ;       The instruction state (stack and general registers) is restored to the
02E0  1697 ;       point where it was when the instruction (routine) was interrupted and
02E0  1698 ;       control is passed back to the top of the control loop or to another
02E0  1699 ;       restart point.
02E0  1700 ;
02E0  1701 ; Input Parameters:
02E0  1702 ;
02E0  1703 ;        31               23               15               07               00
02E0  1704 ;       +----------------+----------------+----------------+----------------+
02E0  1705 ;       |           zero count            |             srclen              | : R0
02E0  1706 ;       +----------------+----------------+----------------+----------------+
02E0  1707 ;       |                              srcaddr                               | : R1
02E0  1708 ;       +----------------+----------------+----------------+----------------+
02E0  1709 ;       | delta-srcaddr  |    delta-PC    |      sign      |      fill      | : R2
02E0  1710 ;       +----------------+----------------+----------------+----------------+
02E0  1711 ;       |                              pattern                               | : R3
02E0  1712 ;       +----------------+----------------+----------------+----------------+
02E0  1713 ;       |   loop-count   |     state      |    saved-PSW   |   inisrclen    | : R4
02E0  1714 ;       +----------------+----------------+----------------+----------------+
02E0  1715 ;       |                              dstaddr                               | : R5
02E0  1716 ;       +----------------+----------------+----------------+----------------+
02E0  1717 ;
02E0  1718 ;       Depending on where the exception occurred, some of these parameters
02E0  1719 ;       may not be relevant. They are nevertheless stored as if they were
02E0  1720 ;       valid to make this restart code as simple as possible.
02E0  1721 ;
02E0  1722 ;       These register fields are more or less architecturally defined. They
02E0  1723 ;       are strictly specified for a reserved operand fault (illegal pattern
02E0  1724 ;       operator) and it makes sense to use the same register fields for
02E0  1725 ;       access violations as well.
02E0  1726 ;
02E0  1727 ;               R0<07:00> - Current digit count in input string
02E0  1728 ;                   (see EO_READ_CHAR below)
02E0  1729 ;               R0<31:16> - Current zero count (loaded into R9)
02E0  1730 ;               R1        - Address of next digit in input string
02E0  1731 ;               R2<07:00> - Fill character
02E0  1732 ;               R2<15:08> - Sign character (loaded into R4)
02E0  1733 ;               R3        - Address of next pattern operator
02E0  1734 ;               R5        - Address of next character in output character string
02E0  1735 ;
02E0  1736 ;       These register fields are specific to this implementation.
02E0  1737 ;
02E0  1738 ;               R0<15:08> - Latest digit from input string (loaded into R7)
02E0  1739 ;               R2<23:16> - Size of instruction (Unused by this routine)
02E0  1740 ;               R2<31:24> - Delta srcaddr (used to compute saved R1)
02E0  1741 ;               R4<07:00> - Initial digit count (stored in saved R0)
02E0  1742 ;               R4<15:08> - Saved condition codes (stored in R11)
02E0  1743 ;                       PSL<N> - Source string has a minus sign
02E0  1744 ;                       PSL<Z> - All digits are zero so far
02E0  1745 ;                       PSL<V> - Nonzero digits have been lost
02E0  1746 ;                       PSL<C> - Significance
02E0  1747 ;               R4<23:16> - State flags
```

```
                     02E0  1748 ;                          State field determines the restart point
                     02E0  1749 ;                  R4<31:24> - Loop count (loaded into R8)
                     02E0  1750 ;
                     02E0  1751 ;       00(SP) - Return PC from VAX$EDITPC routine
                     02E0  1752 ;
                     02E0  1753 ; Implicit Input:
                     02E0  1754 ;
                     02E0  1755 ;       Note that the initial "srclen" is checked for legality before any
                     02E0  1756 ;       restartable exception can occur. This means that R0 LEQU 31, which
                     02E0  1757 ;       leaves bits <15:5> free for storing intermediate state. In the case of
                     02E0  1758 ;       an access violation, R0<15:8> is used to store the latest digit read
                     02E0  1759 ;       from the input stream. In the case of an illegal pattern operator,
                     02E0  1760 ;       R0<15:5> are not used so that the architectural requirement that
                     02E0  1761 ;       R0<15:0> contain the current byte count is adhered to.
                     02E0  1762 ;
                     02E0  1763 ; Output Parameters:
                     02E0  1764 ;
                     02E0  1765 ;       All of the registers are loaded, even if some of their contents are
                     02E0  1766 ;       not relevant to the particular point at which the instruction will be
                     02E0  1767 ;       restarted. This makes the output of this routine conditional on a
                     02E0  1768 ;       single thing, namely on whether the restart point is in one of the
                     02E0  1769 ;       pattern-specific routines or in the outer VAX$EDITPC routine. This
                     02E0  1770 ;       comment applies especially to R7 and R8.
                     02E0  1771 ;
                     02E0  1772 ;       R0  - Current digit count in input string
                     02E0  1773 ;       R1  - Address of next digit in input string
                     02E0  1774 ;       R2  - Fill character
                     02E0  1775 ;       R3  - Address of next pattern operator
                     02E0  1776 ;       R4  - Sign character (stored in R2<15:8>)
                     02E0  1777 ;       R5  - Address of next character to be stored in output character string
                     02E0  1778 ;       R6  - Scratch
                     02E0  1779 ;       R7  - Latest digit read from input packed decimal string
                     02E0  1780 ;       R8  - Loop count
                     02E0  1781 ;       R9  - Zero count (stored in R0<31:16>)
                     02E0  1782 ;       R10 - Address of EDITPC_ACCVIO, this module's "condition handler"
                     02E0  1783 ;       R11 - Condition codes
                     02E0  1784 ;
                     02E0  1785 ;       00(SP) - Saved R0
                     02E0  1786 ;       04(SP) - Saved R1
                     02E0  1787 ;       08(SP) - Saved R6
                     02E0  1788 ;       12(SP) - Saved R7
                     02E0  1789 ;       16(SP) - Saved R8
                     02E0  1790 ;       20(SP) - Saved R9
                     02E0  1791 ;       24(SP) - Saved R10
                     02E0  1792 ;       28(SP) - Saved R11
                     02E0  1793 ;       32(SP) - Return PC from VAX$EDITPC routine
                     02E0  1794 ;
                     02E0  1795 ; Side Effects:
                     02E0  1796 ;
                     02E0  1797 ;       R6 is assumed unimportant and is used as a scratch register by this
                     02E0  1798 ;       routine as soon as it is saved.
                     02E0  1799 ;-
                     02E0  1800
                     02E0  1801 VAX$EDITPC_RESTART::
0FFF 8F    BB        02E0  1802         PUSHR   #^M<R0,R1,R2,R3,R4,R5,R6,R7,R8,R9,R10,R11>      ; Save them all
                     02E4  1803         ESTABLISH_HANDLER       EDITPC_ACCVIO   ; Reload R10 with handler address
50   50    9A        02E9  1804         MOVZBL  R0,R0                           ; Clear out R0<31:8>
```

```
                54    09 AE    9A  02EC  1805          MOVZBL  EDITPC_B_SIGN(SP),R4             ; Put "sign" back into R4
                      00    EF  02F0  1806          EXTZV   #EDITPC_V_STATE,-
                      04        02F2  1807                  #EDITPC_S_STATE,-
                56    12 AE       02F3  1808                  EDITPC_B_STATE(SP),R6           ; Put restart code into R6
                                  02F6  1809
                                  02F6  1810 ; The following two values are not used on all restart paths but R7 and R8
                                  02F6  1811 ; are loaded unconditionally to make this routine simpler. The most extreme
                                  02F6  1812 ; example is that R7 gets recalculated below for the EDITPC_1 restart point.
                                  02F6  1813
                57    01 AE    9A  02F6  1814          MOVZBL  EDITPC_B_EO_READ_CHAR(SP),R7    ; Get latest input digit
                58    13 AE    9A  02FA  1815          MOVZBL  EDITPC_B_LOOP_COUNT(SP),R8      ; Restore loop count
                59    02 AE    32  02FE  1816          CVTWL   EDITPC_W_ZERO_COUNT(SP),R9      ; Reset zero count (R9 LSS 0)
                5B    11 AE    9A  0302  1817          MOVZBL  EDITPC_B_SAVED_PSW(SP),R11      ; Restore saved condition codes
                                  0306  1818
                                  0306  1819 ; The next four instructions reconstruct the initial values of "srclen" and
                                  0306  1820 ; "srcaddr" and store them on the stack just above the saved R6. These values
                                  0306  1821 ; will be loaded into R0 and R1 when the instruction completes execution.
                                  0306  1822 ; Note that these two instructions destroy information in the saved copy of
                                  0306  1823 ; R4 so all of that information must be removed before these instructions
                                  0306  1824 ; execute.
                                  0306  1825
          14 AE    0B AE    9A  0306  1826          MOVZBL  EDITPC_B_DELTA_SRCADDR(SP),EDITPC_L_SAVED_R1(SP)
                   14 AE    C3  030B  1827          SUBL3   EDITPC_L_SAVED_R1(SP),-
                   04 AE        030E  1828                  EDITPC_A_SRCADDR(SP),-
                   14 AE        0310  1829                  EDITPC_L_SAVED_R1(SP)
          10 AE    10 AE    9A  0312  1830          MOVZBL  EDITPC_B_INISRCLEN(SP),EDITPC_L_SAVED_R0(SP)
                                  0317  1831
                                  0317  1832 ; The top four longwords are discarded and control is passed to the restart
                                  0317  1833 ; point obtained from the restart PC table. Note that there is an assumption
                                  0317  1834 ; here that the first two restart points are different from the others in that
                                  0317  1835 ; they do not have an additional return PC (TOP_OF_LOOP) on the stack.
                                  0317  1836
             5E    10    C0     0317  1837          ADDL    #EDITPC_L_SAVED_R0,SP   ; Make saved registers R0, R1, R6, ...
             01    56    D1     031A  1838          CMPL    R6,#EDITPC_1_RESTART    ; Check for restart in main routine
                   06    1B     031D  1839          BLEQU   10$                     ; Branch if no return PC
             FD2D  CF    9F     031F  1840          PUSHAB  TOP_OF_LOOP             ; Restart in some subroutine
                   08    11     0323  1841          BRB     20$                     ; Use common code to resume execution
                                  0325  1842
                                  0325  1843 ; EDITPC_1 is a restart point where R7 must contain the address of the byte
                                  0325  1844 ; that contains the sign "digit". This address must be recalculated. Note that
                                  0325  1845 ; this calculation overwrites the previous R7 restoration.
                                  0325  1846
    57    50    04    01    EF    0325  1847 10$:    EXTZV   #1,#4,R0,R7             ; Get byte offset to end of string
                57    51    C0    032A  1848          ADDL    R1,R7                   ; Get address of byte containing sign
                                  032D  1849
       56    FFFE'CF46    3C     032D  1850 20$:    MOVZWL  RESTART_PC_TABLE_BASE-2[R6],R6  ; Convert code to PC offset
             FCC8 CF46    17     0333  1851          JMP     MODULE_BASE[R6]         ; Get back to work
                                  0338  1852
                                  0338  1853          END_MARK_POINT          EDITPC_M_STATE
                                  0338  1854
                                  0338  1855          .END
```

```
...PC...                      = 00000169
...RESTART_PC...              = 00000174
ADJUST_INPUT_1                  00000298 R     02
BLANK                         = 00000020
BLANK_ZERO_1                    00000298 R     02
BLANK_ZERO_2                    00000279 R     02
BLANK_ZERO_2_RESTART          = 0000000B
EDITPC_1                        00000289 R     02
EDITPC_1_RESTART              = 00000001
EDITPC_2                        0000029B R     02
EDITPC_2_RESTART              = 00000002
EDITPC_3                        00000298 R     02
EDITPC_4                        00000298 R     02
EDITPC_5                        00000298 R     02
EDITPC_ACCVIO                   000001FD R     02
EDITPC_A_PATTERN              = 0000000C
EDITPC_A_SRCADDR              = 00000004
EDITPC_B_DELTA_PC             = 0000000A
EDITPC_B_DELTA_SRCADDR        = 0000000B
EDITPC_B_EO_READ_CHAR         = 00000001
EDITPC_B_INISRCLEN            = 00000010
EDITPC_B_LOOP_COUNT           = 00000013
EDITPC_B_SAVED_PSW            = 00000011
EDITPC_B_SIGN                 = 00000009
EDITPC_B_STATE                = 00000012
EDITPC_L_SAVED_R0             = 00000010
EDITPC_L_SAVED_R1             = 00000014
EDITPC_M_FPD                  = 00000010
EDITPC_M_STATE                = 0000000F
EDITPC_PACK                     000002A2 R     02
EDITPC_ROPRAND_ABORT            000001F4 R     02
EDITPC_ROPRAND_FAULT            000001B3 R     02
EDITPC_S_STATE                = 00000004
EDITPC_V_FPD                  = 00000004
EDITPC_V_STATE                = 00000000
EDITPC_W_SRCLEN               = 00000000
EDITPC_W_ZERO_COUNT           = 00000002
END_FLOAT_1                     0000028F R     02
EO$ADJUST_INPUT_ROUTINE         00000169 R     02
EO$BLANK_ZERO_ROUTINE           0000012B R     02
EO$CLEAR_SIGNIF_ROUTINE         00000161 R     02
EO$END_FLOAT_ROUTINE            00000123 R     02
EO$END_ROUTINE                  00000187 R     02
EO$FILL_ROUTINE                 000000CB R     02
EO$FLOAT_ROUTINE                000000FC R     02
EO$INSERT_ROUTINE               000000B9 R     02
EO$LOAD_FILL_ROUTINE            0000014B R     02
EO$LOAD_MINUS_ROUTINE           0000015A R     02
EO$LOAD_PLUS_ROUTINE            00000153 R     02
EO$LOAD_SIGN_ROUTINE            0000014F R     02
EO$MOVE_ROUTINE                 000000D8 R     02
EO$REPLACE_SIGN_ROUTINE         0000013C R     02
EO$SET_SIGNIF_ROUTINE           00000165 R     02
EO$STORE_SIGN_ROUTINE           000000C7 R     02
EO_READ                         00000095 R     02
FILL_1                          00000298 R     02
FILL_2                          00000255 R     02
FILL_2_RESTART                = 00000003
FLOAT_1                         00000298 R     02
FLOAT_2                         00000267 R     02
FLOAT_2_RESTART               = 00000008
FLOAT_3                         0000026D R     02
FLOAT_3_RESTART               = 00000009
FLOAT_4                         00000273 R     02
FLOAT_4_RESTART               = 0000000A
HANDLER_TABLE_BASE              00000000 R     04
INSERT_1                        00000298 R     02
INSERT_2                        00000298 R     02
LOAD_XXXX_1                     00000298 R     02
LOAD_XXXX_2                     00000298 R     02
MINUS                         = 0000002D
MODULE_BASE                   = 00000000 R     02
MODULE_END                    = 00000338 R     02
MOVE_1                          00000298 R     02
MOVE_2                          0000025B R     02
MOVE_2_RESTART                = 00000005
MOVE_3                          00000261 R     02
MOVE_3_RESTART                = 00000006
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$V_C                       = 00000000
PSL$V_N                       = 00000003
PSL$V_V                       = 00000001
PSL$V_Z                       = 00000002
READ_1                          00000231 R     02
READ_2                          00000231 R     02
REPLACE_SIGN_1                  00000298 R     02
REPLACE_SIGN_2                  00000295 R     02
RESTART_PC_TABLE_BASE           00000000 R     05
RESTART_TABLE_SIZE            = 0000000C
STORE_SIGN_1                    00000298 R     02
TABLE_SIZE                    = 0000001B
TOP_OF_LOOP                     00000050 R     02
VAX$EDITPC                      00000006 RG    02
VAX$EDITPC_OVERFLOW             ********   X   00
VAX$EDITPC_RESTART              000002E0 RG    02
VAX$REFLECT_FAULT               ********   X   00
VAX$ROPRAND                     ********   X   00
ZERO                          = 00000030
```

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	00000338 (824.)	02 (2.)	PIC USR CON REL LCL SHR EXE RD NOWRT NOVEC LONG
PC_TABLE	00000036 (54.)	03 (3.)	PIC USR CON REL LCL SHR NOEXE RD NOWRT NOVEC BYTE
HANDLER_TABLE	00000036 (54.)	04 (4.)	PIC USR CON REL LCL SHR NOEXE RD NOWRT NOVEC BYTE
RESTART_PC_TABLE	00000018 (24.)	05 (5.)	PIC USR CON REL LCL SHR NOEXE RD NOWRT NOVEC BYTE

Performance indicators

Phase	Page faults	CPU Time	Elapsed Time
Initialization	10	00:00:00.02	00:00:02.33
Command processing	71	00:00:00.51	00:00:03.36
Pass 1	160	00:00:05.76	00:00:17.88
Symbol table sort	0	00:00:00.22	00:00:00.68
Pass 2	320	00:00:03.65	00:00:10.65
Symbol table output	12	00:00:00.10	00:00:00.26
Psect synopsis output	2	00:00:00.03	00:00:00.17
Cross-reference output	0	00:00:00.00	00:00:00.00
Assembler run totals	575	00:00:10.30	00:00:35.34

The working set limit was 1500 pages.
34222 bytes (67 pages) of virtual memory were used to buffer the intermediate code.
There were 20 pages of symbol table space allocated to hold 158 non-local and 44 local symbols.
1855 source lines were read in Pass 1, producing 23 object records in Pass 2.
20 pages of virtual memory were used to define 17 macros.

Macro library statistics

Macro library name	Macros defined
_255DUA28:[EMULAT.OBJ]VAXMACROS.MLB;1	8
_255DUA28:[SYSLIB]STARLET.MLB;2	5
TOTALS (all libraries)	13

256 GETS were required to define 13 macros.

There were no errors, warnings or information messages.

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

VAXHANDLR
LIS

VAXCVTLP
LIS

VAXDECIML
LIS

VAXEMULAT
LIS

VAXCVTPL
LIS

VAXEDITPC
LIS