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

```
VV      VV    AAAAAA    XX      XX    AAAAAA      SSSSSSSS  HH      HH  PPPPPPPP
VV      VV    AAAAAA    XX      XX    AAAAAA      SSSSSSSS  HH      HH  PPPPPPPP
VV      VV  AA      AA  XX      XX  AA      AA  SS          HH      HH  PP      PP
VV      VV  AA      AA  XX      XX  AA      AA  SS          HH      HH  PP      PP
VV      VV  AA      AA    XX  XX    AA      AA  SS          HH      HH  PP      PP
VV      VV  AA      AA    XX  XX    AA      AA  SS          HH      HH  PP      PP
VV      VV  AA      AA      XX      AA      AA    SSSSSS    HHHHHHHHHH  PPPPPPPP
VV      VV  AA      AA      XX      AA      AA    SSSSSS    HHHHHHHHHH  PPPPPPPP
VV      VV  AAAAAAAAAA    XX  XX    AAAAAAAAAA          SS  HH      HH  PP
VV      VV  AAAAAAAAAA    XX  XX    AAAAAAAAAA          SS  HH      HH  PP
  VV  VV    AA      AA  XX      XX  AA      AA          SS  HH      HH  PP                ....
  VV  VV    AA      AA  XX      XX  AA      AA          SS  HH      HH  PP                ....
    VV      AA      AA  XX      XX  AA      AA  SSSSSSSS    HH      HH  PP                ....
    VV      AA      AA  XX      XX  AA      AA  SSSSSSSS    HH      HH  PP                ....

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

Table of contents

```
0000            .TITLE  VAX$ASHP - VAX-11 Instruction Emulator for ASHP
0000            .IDENT  /V04-000/
0000
0000    ;
```


```
0000    ;
0000
0000    ;++
0000    ; Facility:
0000    ;
0000    ;       VAX-11 Instruction Emulator
0000    ;
0000    ; Abstract:
0000    ;
0000    ;       The routine in this module emulates the VAX-11 packed decimal
0000    ;       ASHP instruction. This procedure can be a part of an emulator
0000    ;       package or can be called directly after the input parameters
0000    ;       have been loaded into the architectural registers.
0000    ;
0000    ;       The input parameters to this routine are the registers that
0000    ;       contain the intermediate instruction state.
0000    ;
0000    ; Environment:
0000    ;
0000    ;       This routine runs at any access mode, at any IPL, and is AST
0000    ;       reentrant.
0000    ;
0000    ; Author:
0000    ;
0000    ;       Lawrence J. Kenah
0000    ;
0000    ; Creation Date
0000    ;
0000    ;       18 October 1983
0000    ;
0000    ; Modified by:
```

```
0000     ;
0000     ;       V01-003 LJK0042         Lawrence J. Kenah       26-Jul-1984
0000     ;               Final cleanup pass for source kit.
0000     ;
0000     ;       V01-002 LJK0024         Lawrence J. Kenah       20-Feb-1984
0000     ;               Add code to handle access violations. Perform minor cleanup.
0000     ;
0000     ;       V01-001 LJK0008         Lawrence J. Kenah       18-Oct-1983
0000     ;               The emulation code for ASHP was moved into a separate module.
0000     ;--
```

```
    0000          .SUBTITLE       Declarations
    0000
    0000  ; Include files:
    0000
    0000          .NOCROSS                        ; No cross reference for these
    0000          .ENABLE         SUPPRESSION     ; No symbol table entries either
    0000
    0000          ASHP_DEF                        ; Bit fields in ASHP registers
    0000
    0000          ADDP4_DEF                       ; Bit fields in ADDP4 registers
    0000          ADDP6_DEF                       ; Bit fields in ADDP6 registers
    0000          DIVP_DEF                        ; Bit fields in DIVP registers
    0000          MULP_DEF                        ; Bit fields in MULP registers
    0000          SUBP4_DEF                       ; Bit fields in SUBP4 registers
    0000          SUBP6_DEF                       ; Bit fields in SUBP6 registers
    0000
    0000          CVTPS_DEF                       ; Bit fields in CVTPS registers
    0000          CVTPT_DEF                       ; Bit fields in CVTPT registers
    0000          CVTSP_DEF                       ; Bit fields in CVTSP registers
    0000          CVTTP_DEF                       ; Bit fields in CVTTP registers
    0000
    0000          PACK_DEF                        ; Stack usage for reflecting exceptions
    0000          STACK_DEF                       ; Stack usage for original exception
    0000
    0000          $PSLDEF                         ; Define bit fields in PSL
    0000          $SRMDEF                         ; Define arithmetic trap codes
    0000
    0000          .DISABLE        SUPPRESSION     ; Turn on symbol table again
    0000          .CROSS                          ; Cross reference is OK now
    0000
    0000  ; External declarations
    0000
    0000          .DISABLE        GLOBAL
    0000
    0000          .EXTERNAL -
    0000                          DECIMAL$STRIP_ZEROS_R0_R1
    0000
    0000          .EXTERNAL -
    0000                          VAX$EXIT_EMULATOR,-
    0000                          VAX$ADD_PACKED_BYTE_R6_R7,-
    0000                          VAX$REFLECT_FAULT,-
    0000                          VAX$REFLECT_TRAP,-
    0000                          VAX$ROPRAND
    0000
    0000  ; PSECT Declarations:
    0000
    0000          .DEFAULT        DISPLACEMENT , WORD
    0000
00000000          .PSECT _VAX$CODE PIC, USR, CON, REL, LCL, SHR, EXE, RD, NOWRT, LONG
    0000
    0000          BEGIN_MARK_POINT
```

```
0000          .SUBTITLE      VAX$ASHP - Arithmetic Shift and Round Packed
0000    ;+
0000    ; Functional Description:
0000    ;
0000    ;      The source string specified by the  source  length  and  source  address
0000    ;      operands is scaled by a power of 10 specified by the count operand.  The
0000    ;      destination string specified by the destination length  and  destination
0000    ;      address operands is replaced by the result.
0000    ;
0000    ;      A positive count  operand  effectively  multiplies;  a  negative  count
0000    ;      effectively  divides;  and a zero count just moves and affects condition
0000    ;      codes.  When a negative count is specified, the result is rounded  using
0000    ;      the Round Operand.
0000    ;
0000    ; Input Parameters:
0000    ;
0000    ;      R0<15:0>  = srclen.rw   Number of digits in source character string
0000    ;      R0<23:16> = cnt.rb      Shift count
0000    ;      R1        = srcaddr.ab  Address of input character string
0000    ;      R2<15:0>  = dstlen.rw   Length in digits of output decimal string
0000    ;      R2<23:16> = round.rb    Round operand used with negative shift count
0000    ;      R3        = dstaddr.ab  Address of destination packed decimal string
0000    ;
0000    ; Output Parameters:
0000    ;
0000    ;      R0 = 0
0000    ;      R1 = Address of byte containing most significant digit of
0000    ;           the source string
0000    ;      R2 = 0
0000    ;      R3 = Address of byte containing most significant digit of
0000    ;           the destination string
0000    ;
0000    ; Condition Codes:
0000    ;
0000    ;      N <- destination string LSS 0
0000    ;      Z <- destination string EQL 0
0000    ;      V <- decimal overflow
0000    ;      C <- 0
0000    ;
0000    ; Algorithm:
0000    ;
0000    ;      The routine tries as much as possible to work with entire bytes. This
0000    ;      makes the case of an odd shift count more difficult that of an even
0000    ;      shift count. The first part of the routine reduces the case of an odd
0000    ;      shift count to an equivalent operation with an even shift count.
0000    ;
0000    ;      The instruction proceeds in several stages. In the first stage, after
0000    ;      the input parameters have been verified and stored, the operation is
0000    ;      broken up into four cases, based on the sign and parity (odd or even)
0000    ;      of the shift count. These four cases are treated as follows, in order
0000    ;      of increasing complexity.
0000    ;
0000    ;      Case 1. Shift count is negative and even
0000    ;
0000    ;          The actual shift operation can work with the source string in
0000    ;          place. There is no need to move the source string to an
0000    ;          intermediate work area.
```

```
                              0000  ;
                              0000  ;       Case 2. Shift count is positive and even
                              0000  ;
                              0000  ;           The source string is moved to an intermediate work area and the
                              0000  ;           sign "digit" is cleared before the actual shift operation takes
                              0000  ;           place. If the source is worked on in place, then a spurious sign
                              0000  ;           digit would be moved to the middle of the output string instead
                              0000  ;           of a zero. The alternative is to keep track of where, in the
                              0000  ;           several special cases of shifting, the sign digit is looked at.
                              0000  ;           We chose to use the work area to simplify the later stages of
                              0000  ;           this routine.
                              0000  ;
                              0000  ;       Cases 3 and 4. Shift count is odd
                              0000  ;
                              0000  ;           The case of an odd shift count is considerably more difficult
                              0000  ;           than an even shift count, which is only slightly more complicated
                              0000  ;           than MOVP. In the case of an even shift count, various digits
                              0000  ;           remain in the same place (high nibble or low nibble) in a byte.
                              0000  ;           For odd shift counts, high nibbles become low nibbles and vice
                              0000  ;           versa. In addition, digits that were adjacent when viewing the
                              0000  ;           decimal string as a string of bits proceeding from low address to
                              0000  ;           high are now separated by a full byte.
                              0000  ;
                              0000  ;           We proceed in two steps. The source string is first moved to a
                              0000  ;           work area. The string is then shifted by one. This shift reduces
                              0000  ;           the operation to one of the two even shift counts already
                              0000  ;           mentioned, where the source to the shift operation is the
                              0000  ;           modified source string residing in the work area. The details of
                              0000  ;           the shift-by-one are described below near the code that performs
                              0000  ;           the actual shift.
                              0000  ;-
                              0000
              F0F0F0F0        0000          ASHP_SHIFT_MASK = ^XF0F0F0F0    ; Mask used to shift string by one
                              0000
                              0000  VAX$ASHP::
            0FFF 8F    BB     0000          PUSHR   #^M<R0,R1,R2,R3,R4,R5,R6,R7,R8,R9,R10,R11>      ; Save the lot
                 5B    DC     0004          MOVPSL  R11                     ; Get initial PSL
5B    04    00   04    F0     0006          INSV    #PSL$M_Z,#0,#4,R11      ; Set Z-bit, clear the rest
                              000B          ESTABLISH_HANDLER       -       ; Store address of access
                              000B                  ASHP_ACCVIO             ;  violation handler
                              0010          ROPRAND_CHECK   R2              ; Insure that R2 LEQU 31
                              001B          ROPRAND_CHECK   R0              ; Insure that R0 LEQU 31
            58   5E    D0     0023          MOVL    SP,R8                   ; Remember current top of stack
            5E   14    C2     0026          SUBL    #20,SP                  ; Allocate work area on stack
                              0029          MARK_POINT      ASHP_BSBW_20
              FFD4'    30     0029          BSBW    DECIMAL$STRIP_ZEROS_R0_R1       ; Eliminate any high order zeros
52    52    04   01    EF     002C          EXTZV   #1,#4,R2,R2             ; Convert output digit count to bytes
                 52    D6     0031          INCL    R2                      ; Make room for sign as well
50    50    04   01    EF     0033          EXTZV   #1,#4,R0,R0             ; Same for input string
      56    51   50    C1     0038          ADDL3   R0,R1,R6                ; Get address of sign digit
                 50    D6     003C          INCL    R0                      ; Include byte containing sign
                              003E          MARK_POINT      ASHP_20
 56    66     F0 8F    8B     003E          BICB3   #^B11110000,(R6),R6     ; Extract sign digit
                              0043
                              0043  ; Form sign of output string in R9 in preferred form (12 for "+" and 13 for "-")
                              0043
            59   0C    9A     0043          MOVZBL  #12,R9                  ; Assume that input sign is plus
```

```
                                0046          CASE    R6,LIMIT=#10,TYPE=B,<-   ; Dispatch on sign
                                0046                  40$,-                    ; 10 => +
                                0046                  30$,-                    ; 11 => -
                                0046                  40$,-                    ; 12 => +
                                0046                  30$,-                    ; 13 => -
                                0046                  40$,-                    ; 14 => +
                                0046                  40$,-                    ; 15 => +
                                0046                  >
                                0056
                  59     D6     0056    30$:  INCL    R9                       ; Change preferred plus to minus
             5B   08     88     0058          BISB    #PSL$M_N,R11             ; Set N-bit in saved PSW
                                005B
                                005B    ; We now retrieve the shift count from the saved R0 and perform the next set
                                005B    ; of steps based on the parity and sign of the shift count. Note that the
                                005B    ; round operand is ignored unless the shift count is strictly less than zero.
                                005B
        54     02 A8     98     005B    40$:  CVTBL   ASHP_B_CNT(R8),R4        ; Extract sign-extended shift count
                  0D     19     005F          BLSS    50$                      ; Branch if shift count negative
                  55     D4     0061          CLRL    R5                       ; Ignore "round" for positive shift
                0151     30     0063          BSBW    ASHP_COPY_SOURCE         ; Move source string to work area
               11 54     E8     0066          BLBS    R4,60$                   ; Do shift by one for odd shift count
             78   0F     8A     0069          BICB2   #^B00001111,-(R8)        ; Drop sign in saved source string
                  3C     11     006C          BRB     ASHP_SHIFT_POSITIVE      ; Go do the actual shift
                                006E
                                006E    ; The "round" operand is important for negative shifts. If the shift count
                                006E    ; is even, the source can be shifted directly into the destination. For odd
                                006E    ; shift counts, the source must be moved into the work area on the stack and
                                006E    ; shifted by one before the rest of the shift operation takes place.
                                006E
                  00     EF     006E    50$:  EXTZV   #ASHP_V_ROUND,-
                  04            0070                  #ASHP_S_ROUND,-
        55     0A A8            0071                  ASHP_B_ROUND(R8),R5      ; Store "round" in a safe place
               54 54     E9     0074          BLBC    R4,ASHP_SHIFT_NEGATIVE   ; Get right to it for even shift count
                013D     30     0077          BSBW    ASHP_COPY_SOURCE         ; Move source string to work area
                                007A
                                007A    ; For odd shift counts, the saved source string is shifted by one in place.
                                007A    ; This is equivalent to a shift of -1 so the shift count (R4) is adjusted
                                007A    ; accordingly. The least significant digit is moved to the place occupied by
                                007A    ; the sign, the tens digit becomes the units digit, and so on. Because the
                                007A    ; work area was padded with zeros, this shift moves a zero into the high
                                007A    ; order digit of a source string of even length.
                                007A
                  50     DD     007A    60$:  PUSHL   R0                       ; We need a scratch register to count
                  50     D7     007C          DECL    R0                       ; Want to map {1..16} onto {0..3}
   50     50   FE 8F     78     007E          ASHL    #-2,R0,R0                ; Convert a byte count to longwords
                                0083
                                0083    ; The following loop executes from one to four times such that the entire
                                0083    ; source, taken as a collection of longwords, is shifted by one. Note that
                                0083    ; the two pieces of the source are shifted (rotated) in opposite directions.
                                0083    ; Note also that the shift mask is applied to one string before the shift and
                                0083    ; to the other string after the shift. (This points up the arbitrary choice
                                0083    ; of shift mask. We just as well could have chosen the one's complement of
                                0083    ; the shift mask and reversed the order of the shift and mask operations for
                                0083    ; the two pieces of the source string.)
                                0083
   56     78   FC 8F     9C     0083    70$:  ROTL    #-4,-(R8),R6             ; Shift left one digit
   56   F0F0F0F0 8F      CA     0088          BICL2   #ASHP_SHIFT_MASK,R6      ; Clear out old low order digits
```

```
57   FF A8   F0F0F0F0 8F   CB  008F        BICL3   #ASHP_SHIFT_MASK,-1(R8),R7      ; Clear out high order digits
          57      57  04   9C  0098        ROTL    #4,R7,R7                        ; Shift these digits right one digit
          68      57  56   C9  009C        BISL3   R6,R7,(R8)                      ; Combine the two sets of digits
                  E0  50   F4  00A0        SOBGEQ  R0,70$                          ; Keep going if more
                               00A3
                  50  8E   D0  00A3        MOVL    (SP)+,R0                        ; Restore source string byte count
                      54   D6  00A6        INCL    R4                              ; Count the shift we did
                      21   19  00A8        BLSS    ASHP_SHIFT_NEGATIVE             ; Join common code at the right place
                               00AA                                                ; Drop through to ASHP_SHIFT_POSITIVE
```

```
                         00AA          .SUBTITLE       ASHP_SHIFT - Perform Shift
                         00AA    ;+
                         00AA    ; Functional Description:
                         00AA    ;
                         00AA    ;       This routine completes the work of the ASHP instruction in the case of
                         00AA    ;       an even shift count. (If the original shift count was odd, the source
                         00AA    ;       string has already been shifted by one and the shift count adjusted by
                         00AA    ;       one.) A portion (from none to all) of the source string is moved to
                         00AA    ;       the destination string. Pieces of the destination string at either end
                         00AA    ;       may be filled with zeros. If excess digits of the source are not
                         00AA    ;       moved, they must be tested for nonzero to determine the correct
                         00AA    ;       setting of the V-bit.
                         00AA    ;
                         00AA    ; Input Parameters:
                         00AA    ;
                         00AA    ;       R0<3:0> - Number of bytes in source string
                         00AA    ;       R1      - Address of source string
                         00AA    ;       R2<3:0> - Number of bytes in destination string
                         00AA    ;       R3      - Address of destination string
                         00AA    ;       R4<7:0> - Count operand (signed longword of digit count)
                         00AA    ;       R5<3:0> - Round operand in case of negative shift
                         00AA    ;       R9<3:0> - Sign of source string in preferred form
                         00AA    ;
                         00AA    ; Implicit Input:
                         00AA    ;
                         00AA    ;       R4 is presumed (guaranteed) even on input to this routine
                         00AA    ;
                         00AA    ;       The top of the stack is assumed to contain a 20-byte work area (that
                         00AA    ;       may or may not have been used). The space must be allocated for this
                         00AA    ;       work area in all cases so that the exit code works correctly for all
                         00AA    ;       cases without the need for lots of extra conditional code.
                         00AA    ;
                         00AA    ; Output Parameters:
                         00AA    ;
                         00AA    ;       This routine completes the operation of VAX$ASHP. See the routine
                         00AA    ;       header for VAX$ASHP for details on output registers and conditon codes.
                         00AA    ;
                         00AA    ; Details:
                         00AA    ;
                         00AA    ;       PUT SOME OF THE STUFF FROM ASHP.TXT HERE.
                         00AA    ;-
                         00AA
                         00AA          .ENABLE         LOCAL_BLOCK
                         00AA
                         00AA    ASHP_SHIFT_POSITIVE:
        54    02    C6   00AA          DIVL    #2,R4                   ; Convert digit count to byte count
  57    52    54    C3   00AD          SUBL3   R4,R2,R7                ; Modify the destination count
        0E    19         00B1          BLSS    30$                     ; Branch if simply moving zeros
                         00B3
        56    54    D0   00B3          MOVL    R4,R6                   ; Number of zeros at low order end
  58    57    50    C3   00B6    10$:  SUBL3   R0,R7,R8                ; Are there any excess high order digits?
        26    19         00BA          BLSS    60$                     ; No, excess is in source.
                         00BC
                         00BC    ; We only move "srclen" source bytes. The rest of the destination string is
                         00BC    ; filled with zeros.
                         00BC
        57    50    D0   00BC          MOVL    R0,R7                   ; Get number of bytes to actually move
```

```
            3E  11  00BF          BRB     100$                   ; ... and go move them
                    00C1
                    00C1  ; The count argument is larger than the destination length. All of the source
                    00C1  ; is checked for nonzero (overflow check). All of the destination is filled
                    00C1  ; with zeros.
                    00C1
        56  52  D0  00C1  30$:    MOVL    R2,R6                  ; Number of low order zeros
            57  D4  00C4          CLRL    R7                     ; The source string is untouched
        58  50  D0  00C6          MOVL    R0,R8                  ; Number of source bytes to check
            27  11  00C9          BRB     80$                    ; Go do the actual work
                    00CB
                    00CB  ; If the count is negative, then there is no need to fill in low order zeros
                    00CB  ; (R6 is zero). The following code is similar to the above cases, differing
                    00CB  ; in the roles played by source length (R0) and destination length (R2) and
                    00CB  ; also in the first loop (zero fill or overflow check) that executes.
                    00CB
                    00CB  ASHP_SHIFT_NEGATIVE:
            56  D4  00CB          CLRL    R6                     ; No zero fill at low end of destination
        54  54  CE  00CD          MNEGL   R4,R4                  ; Get absolute value of count
        54  02  C6  00D0          DIVL    #2,R4                  ; Convert digit count to byte count
    57  50  54  C3  00D3          SUBL3   R4,R0,R7               ; Get modified source length
            0E  19  00D7          BLSS    70$                    ; Branch if count is larger
                    00D9
    58  52  57  C3  00D9          SUBL3   R7,R2,R8               ; Are there zeros at high end?
            20  18  00DD          BGEQ    100$                   ; Exit to zero fill loop if yes
                    00DF
                    00DF  ; The modified source length is larger than the destination length. Part
                    00DF  ; of the source is moved. The rest is checked for nonzero.
                    00DF
        57  52  D0  00DF          MOVL    R2,R7                  ; Only move "dstlen" bytes
                    00E2
                    00E2  ; In these cases, some digits in the source string will not be moved. If any
                    00E2  ; of these digits is nonzero, then the V-bit must be set.
                    00E2
        58  58  CE  00E2  60$:    MNEGL   R8,R8                  ; Number of bytes in source to check
            0B  11  00E5          BRB     80$                    ; Exit to overflow check loop
                    00E7
                    00E7  ; The count argument is larger than the source length. All of the destination
                    00E7  ; is filled with zeros. The source is ignored.
                    00E7
            57  D4  00E7  70$:    CLRL    R7                     ; No source bytes get moved
        58  52  D0  00E9          MOVL    R2,R8                  ; All of the destination is filled
            11  11  00EC          BRB     100$                   ; Join the zero fill loop
                    00EE
                    00EE  ;+
                    00EE  ; At this point, the three separate counts have all been calculated. Each
                    00EE  ; loop is executed in turn, stepping through the source and destination
                    00EE  ; strings, either alone or in step as appropriate.
                    00EE  ;
                    00EE  ;       R6 - Number of low order digits to fill with zero
                    00EE  ;       R7 - Number of bytes to move intact from source to destination
                    00EE  ;       R8 - Number of excess digits in one or the other string.
                    00EE  ;
                    00EE  ;       If excess source digits, they must be tested for nonzero to
                    00EE  ;       correctly set the V-bit.
                    00EE  ;
                    00EE  ;       If excess destination bytes, they must be filled with zero.
```

```
                    00EE    ;-
                    00EE
                    00EE    ; Test excess source digits for nonzero
                    00EE
                    00EE            MARK_POINT      ASHP_20
         81    95   00EE    75$     TSTB    (R1)+                   ; Is next byte nonzero
         05    12   00F0            BNEQ    90$                     ; Handle overflow out of line
      F9 58    F4   00F2    80$:    SOBGEQ  R8,75$                  ; Otherwise, keep on looking
                    00F5
         11    11   00F5            BRB     120$                    ; Join top of second loop
                    00F7
   5B    02    88   00F7    90$:    BISB    #PSL$M_V,R11            ; Set saved V-bit
   51    58    C0   00FA            ADDL    R8,R1                   ; Skip past rest of excess
         09    11   00FD            BRB     120$                    ; Join top of second loop
                    00FF
                    00FF    ; In this case, the excess digits are found in the destination string. They
                    00FF    ; must be filled with zero.
                    00FF
         58    D5   00FF    100$:   TSTL    R8                      ; Is there really something to do?
         05    13   0101            BEQL    120$                    ; Skip first loop if nothing
                    0103
                    0103            MARK_POINT      ASHP_20
         83    94   0103    110$:   CLRB    (R3)+                   ; Store another zero
      FB 58    F5   0105            SOBGTR  R8,110$                 ; ... and keep on looping
                    0108
                    0108    ; The next loop is where something interesting happens, namely that parts of
                    0108    ; the source string are moved to the destination string. Note that the use of
                    0108    ; bytes rather than digits in this operation makes the detection of nonzero
                    0108    ; digits difficult because the presence of a nonzero digit in the place
                    0108    ; occupied by the sign or in the high order nibble of an even length output
                    0108    ; string and nowhere else would cause the Z-bit to be incorrectly cleared.
                    0108    ; For this reason, we ignore the Z-bit here and make a special pass over the
                    0108    ; output string after all of the special cases have been dealt with. The
                    0108    ; extra overhead of a second trip to memory is offset by the simplicity in
                    0108    ; other places in this routine.
                    0108
         57    D5   0108    120$:   TSTL    R7                      ; Something to do here?
         06    13   010A            BEQL    140$                    ; Skip this loop if nothing
                    010C
                    010C            MARK_POINT      ASHP_20
   83    81    90   010C    130$:   MOVB    (R1)+,(R3)+             ; Move the next byte
      FA 57    F5   010F            SOBGTR  R7,130$                 ; ... and keep on looping
                    0112
                    0112    ; The final loop occurs in some cases of positive shift count where the low
                    0112    ; order digits of the destination must be filled with zeros.
                    0112
         56    D5   0112    140$:   TSTL    R6                      ; Something to do here?
         05    13   0114            BEQL    160$                    ; Skip if loop count is zero
                    0116
                    0116            MARK_POINT      ASHP_20
         83    94   0116    150$:   CLRB    (R3)+                   ; Store another zero
      FB 56    F5   0118            SOBGTR  R6,150$                 ; ... until we're done
                    011B
                    011B    ;+
                    011B    ; At this point, the destination string is complete except for the sign.
                    011B    ; If there is a round operand, that must be added to the destination string.
                    011B    ;
```

```
                        011B  ;             R3 - Address one byte beyond destination string
                        011B  ;             R5 - Round operand
                        011B  ;-
                        011B
             5E  14  C0 011B  160$:  ADDL    #20,SP                          ; Deallocate work area
52   08 AE   04  01  EF 011E         EXTZV   #1,#4,ASHP_W_DSTLEN(SP),R2      ; Get original destination byte coun
             7E  52  7D 0124         MOVQ    R2,-(SP)                        ; Save address and count for Z-bit loop
             58  55  9A 0127         MOVZBL  R5,R8                           ; Load round into carry register
                 15  13 012A         BEQL    180$                            ; Skip next mess unless "round" exists
                        012C
             55  53  D0 012C         MOVL    R3,R5                           ; R5 tracks the addition output
                 56  D4 012F         CLRL    R6                              ; We only need one term and carry in sum
                        0131
                        0131         MARK_POINT      ASHP_8
             57  73  9A 0131  170$:  MOVZBL  -(R3),R7                        ; Get next digit
                        0134         MARK_POINT      ASHP_BSBW_8
              FEC9'  30 0134         BSBW    VAX$ADD_PACKED_BYTE_R6_R7               ; Perform the addition
                 58  D5 0137         TSTL    R8                              ; See if this add produced a carry
                 06  13 0139         BEQL    180$                            ; All done if no more carry
              F3 52  F4 013B         SOBGEQ  R2,170$                         ; Back for the next byte
                        013E
                        013E  ; If we drop through the end of the loop, then the final add produced a carry.
                        013E  ; This must be reflected by setting the V-bit in the saved PSW.
                        013E
             5B  02  88 013E         BISB    #PSL$M_V,R11                    ; Set the saved V-bit
                        0141
                        0141  ; All of the digits are now loaded into the destination string. The condition
                        0141  ; codes, except for the Z-bit, have their correct settings. The sign must be
                        0141  ; set, a check must be made for even digit count in the output string, and
                        0141  ; the various special cases (negative zero, decimal overflow trap, ans so on)
                        0141  ; must be checked before completing the routine.
                        0141
                        0141  ; This entire routine worked with entire bytes, ignoring whether digit counts
                        0141  ; were odd or even. An illegal digit in the upper nibble of an even input string
                        0141  ; is ignored. A nonzero digit in the upper nibble of an even output string is
                        0141  ; not allowed but must be checked for. If one exists, it indicates overflow.
                        0141
              10 AE  E8 0141  180$:  BLBS    <8+ASHP_W_DSTLEN>(SP),-
                 0F     0144                 185$                            ; Skip next if output digit count is odd
                        0145         MARK_POINT      ASHP_8
              F0 8F  93 0145         BITB    #^B11110000,-
              14 BE     0148                 @<8+ASHP_A_DSTADDR>(SP) ; Is most significant digit nonzero?
                 08  13 014A         BEQL    185$                            ; Nothing to worry about if zero
                        014C         MARK_POINT      ASHP_8
              F0 8F  8A 014C         BICB    #^B11110000,-
              14 BE     014F                 @<8+ASHP_A_DSTADDR>(SP) ; Make the digit zero
             5B  02  88 0151         BISB    #PSL$M_V,R11                    ; ... and set the overflow bit
                        0154
                        0154  ; We have not tested for nonzero digits in the output string. This test is
                        0154  ; made by making another pass over the ouptut string. Note that the low
                        0154  ; order digit is unconditionally checked.
                        0154
             52  6E  7D 0154  185$:  MOVQ    (SP),R2                         ; Get address and count
                        0157         MARK_POINT      ASHP_8
         73   F0 8F  93 0157         BITB    #^B11110000,-(R3)               ; Do not test sign in low order byte
                 19  12 015B         BNEQ    187$                            ; Skip loop if nonzero
                 04  11 015D         BRB     186$                            ; Start at bottom of loop
```

```
                          015F
                          015F               MARK_POINT      ASHP_8
                 73   95  015F  183$:   TSTB    -(R3)                   ; Is next higher byte nonzero?
                 13   12  0161          BNEQ    187$                    ; Exit loop if yes
              F9 52   F4  0163  186$:   SOBGEQ  R2,183$                 ; Keep looking for nonzero if more bytes
                          0166
                          0166  ; The entire output string has been scanned and contains no nonzero
                          0166  ; digits. The Z-bit retains its original setting, which is set. If the
                          0166  ; N-bit is also set, then the negative zero must be changed to positive
                          0166  ; zero (unless the V-bit is also set). Note that in the case of overflow,
                          0166  ; the N-bit is cleared but the output string retrins the minus sign.
                          0166
         0F 5B    03   E1  0166          BBC     #PSL$V_N,R11,190$       ; N-bit is off already
            5B    08   8A  016A          BICB    #PSL$M_N,R11            ; Turn off saved N-bit unconditionally
         08 5B    01   E0  016D          BBS     #PSL$V_V,R11,190$       ; No fixup if V-bit is also set
            59    0C   90  0171          MOVB    #12,R9                  ; Use preferred plus as sign of output
                  03   11  0174          BRB     190$                    ; ... and rejoin the exit code
                           0176
                           0176  ; The following instruction is the exit point for all of the nonzero byte
                           0176  ; checks. Its direct effect is to clear the saved Z-bit. It also bypasses
                           0176  ; whatever other zero checks have not yet been performed.
                           0176
            5B    04   8A  0176  187$:   BICB    #PSL$M_Z,R11            ; Clear saved Z-bit
                           0179
                           0179  ; The following code executes in all cases. It is the common exit path for
                           0179  ; all of the ASHP routines when the count is even.
                           0179
            52    8E   7D  0179  190$:   MOVQ    (SP)+,R2                ; Get address of end of output string
                           017C          MARK_POINT      ASHP_0
 FF A3   04   00  59   F0  017C          INSV    R9,#0,#4,-1(R3)         ; Store sign that we have been saving
                           0182
                           0182          .DISABLE        LOCAL_BLOCK
                           0182
                           0182  ; Drop into VAX$DECIMAL_EXIT for final processing
```

```
                         0182        .SUBTITLE       VAX$DECIMAL_EXIT - Exit Path for Decimal Instructions
                         0182  ;+
                         0182  ; This is the common exit path for many of the routines in this module. This
                         0182  ; exit path can only be used for instructions that conform to the following
                         0182  ; restrictions.
                         0182  ;
                         0182  ; 1.  Registers R0 through R11 were saved on entry.
                         0182  ;
                         0182  ; 2.  The architecture requires that R0 and R2 are zero on exit.
                         0182  ;
                         0182  ; 3.  All other registers that have instruction-specific values on exit are
                         0182  ;     correctly stored in the appropriate locations on the stack.
                         0182  ;
                         0182  ; 4.  The saved PSW is contained in R11
                         0182  ;
                         0182  ; 5.  This instruction/routine should generate a decimal overflow trap if
                         0182  ;     both the V-bit and the DV-bit are set on exit.
                         0182  ;-
                         0182
                         0182        .ENABLE         LOCAL_BLOCK
                         0182
                         0182  VAX$DECIMAL_EXIT::
              6E    D4   0182        CLRL    (SP)                    ; R0 must be zero on exit
           08 AE    D4   0184        CLRL    8(SP)                   ; R2 must also be zero
     09 5B    01    E0   0187        BBS     #PSL$V_V,R11,20$        ; See if exceptions are enabled
              0F    B9   018B  10$:  BICPSW  #<PSL$M_N!PSL$M_Z!PSL$M_V!PSL$M_C>      ; Clear condition codes
              5B    B8   018D        BISPSW  R11                     ; Set appropriate condition codes
         0FFF 8F    BA   018F        POPR    #^M<R0,R1,R2,R3,R4,R5,R6,R7,R8,R9,R10,R11>      ; Restore
                    05   0193        RSB                             ; ... and return
                         0194
                         0194  ; If the V-bit is set and decimal traps are enabled (DV-bit is set), then
                         0194  ; a decimal overflow trap is generated. Note that the DV-bit can be set in
                         0194  ; the current PSL or, if this routine was entered as the result of an emulated
                         0194  ; instruction exception, in the saved PSL on the stack. Note that the final
                         0194  ; condition codes in the PSW have not yet been set. This means that all exit
                         0194  ; paths out of this code must set the condition codes to their correct values.
                         0194
                         0194  VAX$EDITPC_OVERFLOW::
     10 5B    07    E0   0194  20$:  BBS     #PSL$V_DV,R11,30$       ; Report exception if current DV-bit set
  50    0000'CF     9E   0198        MOVAB   VAX$EXIT_EMULATOR,R0    ; Set up R0 for PIC address comparison
     30 AE    50    D1   019D        CMPL    R0,<4*12>(SP)           ; Is return PC EQLU VAX$EXIT_EMULATOR ?
              E8    12   01A1        BNEQU   10$                     ; No. Simply return V-bit set
  E3 60 AE    07    E1   01A3        BBC     #PSL$V_DV,<<4*<12+1>>+EXCEPTION_PSL>(SP),10$
                         01A8                                        ; Only return V-bit if DV-bit is clear
                         01A8
                         01A8  ; Restore all of the saved registers, reset the condition codes, and drop
                         01A8  ; into DECIMAL_OVERFLOW.
                         01A8
              0F    B9   01A8  30$:  BICPSW  #<PSL$M_N!PSL$M_Z!PSL$M_V!PSL$M_C>      ; Clear condition codes
              5B    B8   01AA        BISPSW  R11                     ; Set appropriate condition codes
         0FFF 8F    BA   01AC        POPR    #^M<R0,R1,R2,R3,R4,R5,R6,R7,R8,R9,R10,R11>
```

```
                 01B0    ;+
                 01B0    ; This code path is entered if the decimal string result is too large to
                 01B0    ; fit into the output string and decimal overflow exceptions are enabled.
                 01B0    ; The final state of the instruction, including the condition codes, is
                 01B0    ; entirely in place.
                 01B0    ;
                 01B0    ; Input Parameter:
                 01B0    ;
                 01B0    ;       (SP) - Return PC
                 01B0    ;
                 01B0    ; Output Parameters:
                 01B0    ;
                 01B0    ;       0(SP) - SRM$K_DEC_OVF_T (Arithmetic trap code)
                 01B0    ;       4(SP) - Final state PSL
                 01B0    ;       8(SP) - Return PC
                 01B0    ;
                 01B0    ; Implicit Output:
                 01B0    ;
                 01B0    ;       Control passes through this code to VAX$REFLECT_TRAP.
                 01B0    ;-
                 01B0
                 01B0    VAX$DECIMAL_OVERFLOW::
        7E   DC  01B0           MOVPSL  -(SP)                   ; Save final PSL on stack
        06   DD  01B2           PUSHL   #SRM$K_DEC_OVF_T        ; Store arithmetic trap code
     FE49'   31  01B4           BRW     VAX$REFLECT_TRAP        ; Report exception
                 01B7
                 01B7           .DISABLE        LOCAL_BLOCK
```

```
                         01B7          .SUBTITLE       ASHP_COPY_SOURCE - Copy Source String to Work Area
                         01B7   ;+
                         01B7   ; Functional Description:
                         01B7   ;
                         01B7   ;       For certain cases (three out of four), it is necessary to put the
                         01B7   ;       source string in a work area so that later portions of VAX$ASHP can
                         01B7   ;       proceed in a straightforward manner. In one case (positive even shift
                         01B7   ;       count), the sign must be eliminated before the least significant
                         01B7   ;       byte of the source is moved to its appropriate place (not the least
                         01B7   ;       significant byte) in the destination string. For odd shift counts,
                         01B7   ;       the source string in the work area is shifted by one to reduce the
                         01B7   ;       complicated case of an odd shift count to an equivalent but simpler
                         01B7   ;       case with an even shift count.
                         01B7   ;
                         01B7   ;       This routine moves the source string to a 20-byte work area already
                         01B7   ;       allocated on the stack. Note that the work area is zeroed by this
                         01B7   ;       routine so that, if the work area is used, it consists of either
                         01B7   ;       valid bytes from the source string or bytes containing zero. If the
                         01B7   ;       work area is not needed (shift count is even and not positive), the
                         01B7   ;       overhead of zeroing the work area is avoided.
                         01B7   ;
                         01B7   ; Input Parameters:
                         01B7   ;
                         01B7   ;       R0 - Byte count of source string (preserved)
                         01B7   ;       R1 - Address of most significant byte in source string
                         01B7   ;       R8 - Address one byte beyond end of work area (preserved)
                         01B7   ;
                         01B7   ; Output Parameters:
                         01B7   ;
                         01B7   ;       R1 - Address of most significant byte of source string in
                         01B7   ;            work area
                         01B7   ;
                         01B7   ; Side Effects:
                         01B7   ;
                         01B7   ;       R6 and R7 are modified by this routine.
                         01B7   ;-
                         01B7
                         01B7   ASHP_COPY_SOURCE:
       F8 A8       7C    01B7          CLRQ    -8(R8)                  ; Insure that the work area
       F0 A8       7C    01BA          CLRQ    -16(R8)                 ; ... is entirely filled
       EC A8       D4    01BD          CLRL    -20(R8)                 ; ... with zeros
57   51    50      C1    01C0          ADDL3   R0,R1,R7                ; R7 points one byte beyond source
     51    58      D0    01C4          MOVL    R8,R1                   ; R1 will step through work area
     56    50      D0    01C7          MOVL    R0,R6                   ; Use R6 as the loop counter
                         01CA
                         01CA          MARK_POINT      ASHP_BSBW_20
     71    77      90    01CA   10$:   MOVB    -(R7),-(R1)             ; Move the next source byte
       FA  56      F5    01CD          SOBGTR  R6,10$                  ; Check for end of loop
                         01D0
                   05    01D0          RSB                             ; Return with R1 properly loaded
```

```
                  01D1          .SUBTITLE       DECIMAL_ROPRAND
                  01D1   ;-
                  01D1   ; Functional Description:
                  01D1   ;
                  01D1   ;       This routine receives control when a digit count larger than 31 is
                  01D1   ;       detected. The exception is architecturally defined as an abort so
                  01D1   ;       there is no need to store intermediate state. The VAX$ASHP routine
                  01D1   ;       saves all registers R0 through R11 before performing the digit check.
                  01D1   ;       These registers must be restored before control is passed to
                  01D1   ;       VAX$ROPRAND.
                  01D1   ;
                  01D1   ; Input Parameters:
                  01D1   ;
                  01D1   ;       00(SP) - Saved R0
                  01D1   ;          .
                  01D1   ;          .
                  01D1   ;       44(SP) - Saved R11
                  01D1   ;       48(SP) - Return PC from VAX$xxxxxx routine
                  01D1   ;
                  01D1   ; Output Parameters:
                  01D1   ;
                  01D1   ;       00(SP) - Offset in packed register array to delta PC byte
                  01D1   ;       04(SP) - Return PC from VAX$xxxxxx routine
                  01D1   ;
                  01D1   ; Implicit Output:
                  01D1   ;
                  01D1   ;       This routine passes control to VAX$ROPRAND where further
                  01D1   ;       exception processing takes place.
                  01D1   ;-
                  01D1
                  01D1   DECIMAL_ROPRAND:
OFFF 8F     BA    01D1          POPR    #^M<R0,R1,R2,R3,R4,R5,R6,R7,R8,R9,R10,R11>
     03     DD    01D5          PUSHL   #ASHP_B_DELTA_PC         ; Store offset to delta PC byte
  FE26'     31    01D7          BRW     VAX$ROPRAND              ; Pass control along
```

```
01DA            .SUBTITLE       ASHP_ACCVIO - Reflect an Access Violation
01DA     ;+
01DA     ;      The general approach to handling access violations that occur while
01DA     ;      emulating the packed decimal instructions is described here. We take
01DA     ;      advantage of the fact that there are no architectural constraints on
01DA     ;      the way that access violations must be handled. In general, we back
01DA     ;      the instruction up to the beginning, to the point where initial state
01DA     ;      is stored in the set of general registers modified by each
01DA     ;      instruction. Thus, the only step that is avoided when an instruction
01DA     ;      is restarted is operand evaluation.
01DA     ;
01DA     ; Functional Description:
01DA     ;
01DA     ;      This routine (or its counterpart in other decimal emulation modules)
01DA     ;      receives control when an access violation occurs while executing
01DA     ;      within one of the VAX$xxxxxx routines that emulated a decimal string
01DA     ;      instruction. This routine determines whether the exception occurred
01DA     ;      while accessing a source or destination string or whether the access
01DA     ;      violation is peculiar to emulation, such as stack overflow. (This
01DA     ;      check is made based on the PC of the exception.)
01DA     ;
01DA     ;      If the PC is one that is recognized by this routine, then the state of
01DA     ;      the instruction (digit counts, string addresses, and the like) are
01DA     ;      restored to a state where the instruction/routine can be restarted
01DA     ;      after (if) the cause for the exception is eliminated. Control is then
01DA     ;      passed to a common routine that sets up the stack and the exception
01DA     ;      parameters in such a way that the instruction or routine can restart
01DA     ;      transparently.
01DA     ;
01DA     ;      If the exception occurs at some unrecognized PC, then the exception is
01DA     ;      reflected to the user as an exception that occurred within the
01DA     ;      emulator.
01DA     ;
01DA     ;      There are two exceptions that can occur that are not backed up to
01DA     ;      appear as if they occurred at the site of the original emulated
01DA     ;      instruction. These exceptions will appear to the user as if they
01DA     ;      occurred inside the emulator itself.
01DA     ;
01DA     ;          1.  If stack overflow occurs due to use of the stack by one of
01DA     ;              the routines, it is unlikely that this routine will even
01DA     ;              execute because the code that transfers control here must
01DA     ;              first copy the parameters to the exception stack and that
01DA     ;              operation would fail. (The failure causes control to be
01DA     ;              transferred to VMS, where the stack expansion logic is
01DA     ;              invoked and the routine resumed transparently.)
01DA     ;
01DA     ;          2.  If assumptions about the address space change out from under
01DA     ;              these routines (because an AST deleted a portion of the
01DA     ;              address space or a similar silly thing), the handling of the
01DA     ;              exception is UNPREDICTABLE.
01DA     ;
01DA     ; Input Parameters:
01DA     ;
01DA     ;      R0  - Value of SP when exception occurred
01DA     ;      R1  - PC at which exception occurred
01DA     ;      R2  - scratch
01DA     ;      R3  - scratch
```

```
01DA ;       R10 - Address of this routine (no longer needed)
01DA ;
01DA ;       00(SP) - Value of R0 when exception occurred
01DA ;       04(SP) - Value of R1 when exception occurred
01DA ;       08(SP) - Value of R2 when exception occurred
01DA ;       12(SP) - Value of R3 when exception occurred
01DA ;       16(SP) - Return PC in exception dispatcher in operating system
01DA ;
01DA ;       20(SP) - First longword of system-specific exception data
01DA ;         .
01DA ;         .
01DA ;       xx(SP) - Last longword of system-specific exception data
01DA ;
01DA ;       The address of the next longword is the position of the stack when
01DA ;       the exception occurred. R0 locates this address.
01DA ;
01DA ; R0 -> xx+4(SP)      - Instruction-specific data
01DA ;         .           - Optional instruction-specific data
01DA ;         .           - Optional instruction-specific data
01DA ;       xx+<4*M>(SP) - Return PC from VAX$xxxxxx routine (M is the number
01DA ;                      of instruction-specific longwords)
01DA ;
01DA ; Implicit Input:
01DA ;
01DA ;       It is assumed that the contents of all registers coming into this
01DA ;       routine are unchanged from their contents when the exception occurred.
01DA ;       (For R0 through R3, this assumption applies to the saved register
01DA ;       contents on the top of the stack. Any modification to these four
01DA ;       registers must be made to their saved copies and not to the registers
01DA ;       themselves.)
01DA ;
01DA ;       Finally, the macro BEGIN_MARK_POINT should have been invoked at the
01DA ;       beginning of this module to define the symbols
01DA ;
01DA ;               MODULE_BASE
01DA ;               PC_TABLE_BASE
01DA ;               HANDLER_TABLE_BASE
01DA ;               TABLE_SIZE
01DA ;
01DA ;       PC_TABLE_BASE is the base of a word array with one entry for each
01DA ;               PC (relative to MODULE_BASE) that can cause an access
01DA ;               violation that is capable of being backed up.
01DA ;
01DA ;       HANDLER_TABLE_BASE is the base of a corresponding word array with an
01DA ;               entry that locates the context specific code to handle each
01DA ;               of the recognized access violations.
01DA ;
01DA ; Output Parameters (Exit via JMP instruction):
01DA ;
01DA ;       If the exception is recognized (that is, if the exception PC is
01DA ;       associated with one of the mark points), control is passed to the
01DA ;       context-specific routine that restores the instruction state to
01DA ;       its initial state.
01DA ;
01DA ;       These are the register values and stack state when the context
01DA ;       specific code begins execution.
01DA ;
```

```
                          01DA   ;               R0  - Value of SP when exception occurred
                          01DA   ;               R1  - scratch
                          01DA   ;               R2  - scratch
                          01DA   ;               R3  - scratch
                          01DA   ;               R10 - scratch
                          01DA   ;
                          01DA   ; R0 -> zz(SP) - Instruction-specific data begins here
                          01DA   ;
                          01DA   ; Implicit Output:
                          01DA   ;
                          01DA   ;       The context-specific code accomplishes essentially the same thing for
                          01DA   ;       all of the emulated instructions.
                          01DA   ;
                          01DA   ;       The register contents are restored to the values that they had on
                          01DA   ;       entry to the VAX$xxxxxx routine. This causes the instruction to be
                          01DA   ;       backed up almost (but not quite) to its starting point. (The operand
                          01DA   ;       evaluation is not lost. The operands are saved in registers.) Any
                          01DA   ;       registers saved on entry are restored.
                          01DA   ;
                          01DA   ; Output Parameters (Exit via RSB instruction):
                          01DA   ;
                          01DA   ;       If the exception PC occurred somewhere else (such as a stack access),
                          01DA   ;       the saved registers are restored and control is passed back to the
                          01DA   ;       host system with an RSB instruction.
                          01DA   ;
                          01DA   ;-
                          01DA
                          01DA   ASHP_ACCVIO:
              52    D4    01DA           CLRL    R2                      ; Initialize the counter
         FE20 CF    9F    01DC           PUSHAB  MODULE_BASE             ; Store base address of this module
         0244'CF    9F    01E0           PUSHAB  MODULE_END              ; Store module end address
            0020    30    01E4           BSBW    DECIMAL$BOUNDS_CHECK    ; Check if PC is inside the module
         5E   04    C0    01E7           ADDL    #4,SP                   ; Discard end address
         51   8E    C2    01EA           SUBL2   (SP)+,R1                ; Get PC relative to this base
                          01ED
   0000'CF42   51    B1   01ED   10$:    CMPW    R1,PC_TABLE_BASE[R2]    ; Is this the right PC?
              07    13    01F3           BEQL    30$                     ; Exit loop if true
      F4 52   0E    F2    01F5           AOBLSS  #TABLE_SIZE,R2,10$      ; Do the entire table
                          01F9
                          01F9   ; If we drop through the dispatching based on PC, then the exception is not
                          01F9   ; one that we want to back up. We simply reflect the exception to the user.
                          01F9
              0F    BA    01F9   20$:    POPR    #^M<R0,R1,R2,R3>        ; Restore saved registers
                    05    01FB           RSB                             ; Return to exception dispatcher
                          01FC
                          01FC   ; The exception PC matched one of the entries in our PC table. R2 contains
                          01FC   ; the index into both the PC table and the handler table. R1 has served
                          01FC   ; its purpose and can be used as a scratch register.
                          01FC
  51   0000'CF42    3C    01FC   30$:    MOVZWL  HANDLER_TABLE_BASE[R2],R1       ; Get the offset to the handler
       FDF9 CF41    17    0202           JMP     MODULE_BASE[R1]         ; Pass control to the handler
                          0207
                          0207   ; In all of the instruction-specific routines, the state of the stack
                          0207   ; will be shown as it was when the exception occurred. All offsets will
                          0207   ; be pictured relative to R0.
```

```
                        0207          .SUBTITLE       DECIMAL$BOUNDS_CHECK - Bounds Check on Exception PC
                        0207    ;+
                        0207    ; Functional Decsription:
                        0207    ;
                        0207    ;       This routine is called by the exception handlers for the various
                        0207    ;       decimal string instruction emulation routines to perform a bounds
                        0207    ;       check on the exception PC. The real reason for performing this check
                        0207    ;       is that certain exceptions can occur in subroutines that are outside a
                        0207    ;       given module. In this case, it is not the exception PC but rather the
                        0207    ;       return PC on the top of the stack that determines the context of the
                        0207    ;       exception (and therefore, the code necessary to back up the
                        0207    ;       instructic- state).
                        0207    ;
                        0207    ;       The basic mode of operation is that, if the exception PC is outside
                        0207    ;       the current module boundarise, then R1 (the exception PC) is replaced
                        0207    ;       by the return PC (presumed pointed to by R0).
                        0207    ;
                        0207    ; Input Parameters:
                        0207    ;
                        0207    ;       R0 - Top of stack when exception occurred
                        0207    ;       R1 - PC at time of exception
                        0207    ;
                        0207    ;       (R0) - Return PC from subroutine in which access violation occurred
                        0207    ;
                        0207    ;       00(SP) - Return PC from caller of this routine
                        0207    ;       04(SP) - End address of module
                        C207    ;       08(SP) - Start address of module
                        0207    ;
                        0207    ; Output Parameters:
                        0207    ;
                        0207    ;       If the exception PC is outside the bounds of the module (as defined by
                        0207    ;       the two longwords on the stack, then R1 is replaced by the "return
                        0207    ;       PC", the contents of the longword located by R0.
                        0207    ;
                        0207    ;       If the exception PC is inside the module, nothing is changed by the
                        0207    ;       execution of this module.
                        0207    ;
                        0207    ; Assumptions:
                        0207    ;
                        0207    ;       There are two assumptions that must hold for these subroutines
                        0207    ;       in which access violations can occur.
                        0207    ;
                        0207    ;               They must not use the stack. This keeps the return PC on the
                        0207    ;               top of the stack, located by R0.
                        0207    ;
                        0207    ;               They must be called with a BSBW instruction. This causes the
                        0207    ;               return PC to be exactly three bytes beyond instruction that
                        0207    ;               transferred control to the subroutine.
                        0207    ;-
                        0207
                        0207    DECIMAL$BOUNDS_CHECK::
   04 AE    51    D1    0207          CMPL    R1,4(SP)                ; Beyond upper end?
            07    1E    020B          BGEQU   10$                     ; Branch if out of bounds
   08 AE    51    D1    020D          CMPL    R1,8(SP)                ; Within lower limit?
            01    1F    0211          BLSSU   10$                     ; Branch if out of bounds
                  05    0213          RSB                             ; Return with R1 intact
                        0214
```

```
                         0214  ; R1 is out of bounds. Replace it with the return PC from the routine that
                         0214  ; was executing when the access violation occurred. Note that the PC is
                         0214  ; backed up over the BSBW instruction because the PC offset that appears in
                         0214  ; the PC_TABLE will be the PC of the BSBW instruction and not the PC of the
                         0214  ; next instruction.
                         0214
   51    60    03    C3  0214  10$:    SUBL3   #3,(R0),R1              ; Get new "exception" PC
                     05  0218          RSB
```

```
                0219        .SUBTITLE       Context-Specific Access Violation Handling for VAX$ASHP
                0219  ;+
                0219  ; Functional Description:
                0219  ;
                0219  ;       The only difference among the various entry points is the number of
                0219  ;       longwords on the stack. R0 is advanced beyond these longwords to point
                0219  ;       to the list of saved registers. These registers are then restored,
                0219  ;       effectively backing the routine up to its initial state.
                0219  ;
                0219  ; Input Parameters:
                0219  ;
                0219  ;       R0 - Address of top of stack when access violation occurred
                0219  ;
                0219  ;       See specific entry points for details
                0219  ;
                0219  ; Output Parameters:
                0219  ;
                0219  ;       See input parameter list for VAX$DECIMAL_ACCVIO
                0219  ;-
                0219
                0219  ;+
                0219  ; ASHP_BSBW_20
                0219  ;
                0219  ; An access violation occurred in subroutine STRIP_ZEROS or in subroutine
                0219  ; ASHP_COPY_SOURCE while the source string was being copied to the work space
                0219  ; on the stack. In addition to the five longwords of work space on the stack,
                0219  ; this routine has an additional longword, the return PC, on the stack.
                0219  ;
                0219  ;       00(R0) - Return PC in mainline of VAX$ASHP
                0219  ;       04(R0) - First longword of scratch space
                0219  ;        etc.
                0219  ;-
                0219
                0219  ASHP_BSBW_20:
50  04  C0      0219        ADDL    #4,R0                   ; Skip over return PC and drop into ...
                021C
                021C  ;+
                021C  ; ASHP_20
                021C  ;
                021C  ; There are five longwords of workspace on the stack for this entry point.
                021C  ;
                021C  ;       00(R0) - First longword of scratch space
                021C  ;         .
                021C  ;         .
                021C  ;       16(R0) - Fifth longword of scratch space
                021C  ;       20(SP) - Saved R0
                021C  ;       24(SP) - Saved R1
                021C  ;        etc.
                021C  ;-
                021C
                021C  ASHP_20:
50  14  C0      021C        ADDL    #20,R0                  ; Discard scratch space on stack
    06  11      021F        BRB     VAX$DECIMAL_ACCVIO      ; Join common code to restore registers
                0221
                0221  ;+
                0221  ; ASHP_BSBW_8
                0221  ;
```

```
                0221  ; An access violation occurred in subroutine ADD_PACKED_BYTE while the round
                0221  ; operand was being propogated. In addition to the saved R2/R3 pair of
                0221  ; longwords on the stack, this routine has an additional longword, the return
                0221  ; PC, on the stack.
                0221  ;
                0221  ;       00(R0) - Return PC in mainline of VAX$ASHP
                0221  ;       04(R0) - Saved intermediate value of R2
                0221  ;        etc.
                0221  ;-
                0221
                0221  ASHP_BSBW_8:
 50   04   C0   0221          ADDL    #4,R0                   ; Skip over return PC and drop into ...
                0224
                0224  ;+
                0224  ; ASHP_8
                0224  ;
                0224  ; There is a saved register pair (two longwords) on the stack for these entry
                0224  ; points.
                0224  ;
                0224  ;       00(R0) - Saved intermediate value of R2
                0224  ;       04(R0) - Saved intermediate value of R3
                0224  ;       08(SP) - Saved R0
                0224  ;       12(SP) - Saved R1
                0224  ;       16(SP) - Saved R2
                0224  ;       20(SP) - Saved R3
                0224  ;        etc.
                0224  ;-
                0224
                0224  ASHP_8:
 50   08   C0   0224          ADDL    #8,R0                   ; Discard saved register pair
                0227
                0227  ; Drop into VAX$DECIMAL_ACCVIO to restore saved registers
                0227
                0227  ;+
                0227  ; ASHP_0
                0227  ;
                0227  ; The stack is empty. This label is merely a synonym for VAX$DECIMAL_ACCVIO
                0227  ; because there is no context-specific work to do.
                0227  ;
                0227  ;       00(SP) - Saved R0
                0227  ;       04(SP) - Saved R1
                0227  ;       08(SP) - Saved R2
                0227  ;       12(SP) - Saved R3
                0227  ;        etc.
                0227  ;-
                0227
                0227  ASHP_0:
                0227
                0227  ; Drop into VAX$DECIMAL_ACCVIO to restore saved registers
```

```
                    0227          .SUBTITLE       VAX$DECIMAL_ACCVIO - Common Access Violation Handling
                    0227  ;+
                    0227  ; Functional Description:
                    0227  ;
                    0227  ;       This code is the final access violation processing for those
                    0227  ;       exceptions that have two things in common.
                    0227  ;
                    0227  ;       The instruction/routine is to be backed up to its initial state.
                    0227  ;
                    0227  ;       All registers from R0 to R11 were saved on entry to VAX$xxxxxx.
                    0227  ;
                    0227  ; Input Parameters:
                    0227  ;
                    0227  ;       00(R0) - Saved R0 on entry to VAX$xxxxxx
                    0227  ;       04(R0) - Saved R1
                    0227  ;         .
                    0227  ;         .
                    0227  ;       44(R0) - Saved R11 on entry to VAX$xxxxxx
                    0227  ;       48(R0) - Return PC from VAX$xxxxxx routine
                    0227  ;
                    0227  ;       00(SP) - Saved R0 (restored by VAX$HANDLER)
                    0227  ;       04(SP) - Saved R1
                    0227  ;       08(SP) - Saved R2
                    0227  ;       12(SP) - Saved R3
                    0227  ;
                    0227  ; Output Parameters:
                    0227  ;
                    0227  ;       R0 is advanced over saved register array as the registers are restored.
                    0227  ;       R0 ends up pointing at the return PC.
                    0227  ;
                    0227  ;       R1 contains the value of delta PC for all of the routines that
                    0227  ;       use this common code path. The FPD and ACCVIO bits are both set
                    0227  ;       in R1.
                    0227  ;
                    0227  ;       00(R0) - Return PC from VAX$xxxxxx routine
                    0227  ;
                    0227  ;       00(SP) - Value of R0 on entry to VAX$xxxxxx
                    0227  ;       04(SP) - Value of R1 on entry to VAX$xxxxxx
                    0227  ;       08(SP) - Value of R2 on entry to VAX$xxxxxx
                    0227  ;       12(SP) - Value of R3 on entry to VAX$xxxxxx
                    0227  ;
                    0227  ;       R4 through R11 are restored to their values on entry to VAX$xxxxxx.
                    0227  ;-
                    0227
                    0227  VAX$DECIMAL_ACCVIO::
   6E    80    7D   0227          MOVQ    (R0)+,PACK_L_SAVED_R0(SP)        ; "Restore" R0 and R1
08 AE    80    7D   022A          MOVQ    (R0)+,PACK_L_SAVED_R2(SP)        ; "Restore" R2 and R3
   54    80    7D   022E          MOVQ    (R0)+,R4                         ; Really restore R4 and R5
   56    80    7D   0231          MOVQ    (R0)+,R6                         ; ... and R6 and R7
   58    80    7D   0234          MOVQ    (R0)+,R8                         ; ... and R8 and R8
   5A    80    7D   0237          MOVQ    (R0)+,R10                        ; ... and R10 and R11
                    023A
                    023A          ASSUME ADDP4_B_DELTA_PC EQ ASHP_B_DELTA_PC
                    023A          ASSUME ADDP6_B_DELTA_PC EQ ASHP_B_DELTA_PC
                    023A          ASSUME SUBP4_B_DELTA_PC EQ ASHP_B_DELTA_PC
                    023A          ASSUME SUBP6_B_DELTA_PC EQ ASHP_B_DELTA_PC
                    023A          ASSUME MULP_B_DELTA_PC EQ ASHP_B_DELTA_PC
```

```
                          023A         ASSUME DIVP_B_DELTA_PC EQ ASHP_B_DELTA_PC
                          023A
                          023A         ASSUME CVTPS_B_DELTA_PC EQ ASHP_B_DELTA_PC
                          023A         ASSUME CVTPT_B_DELTA_PC EQ ASHP_B_DELTA_PC
                          023A         ASSUME CVTSP_B_DELTA_PC EQ ASHP_B_DELTA_PC
                          023A         ASSUME CVTTP_B_DELTA_PC EQ ASHP_B_DELTA_PC
                          023A
51   00000303 8F     D0   023A         MOVL    #<ASHP_B_DELTA_PC!-      ; Indicate offset for delta PC
                          0241                 PACK_M_FPD!-             ; FPD bit should be set
                          0241                 PACK_M_ACCVIO>,R1        ; This is an access violation
             FDBC'   31   0241         BRW     VAX$REFLECT_FAULT        ; Continue exception handling
                          0244
                          0244         END_MARK_POINT
                          0244
                          0244         .END
```

```
...PC...                        = 000001CA
...ROPRAND...                   = 00000015 R      02
ADDP4_B_DELTA_PC                = 00000003
ADDP6_B_DELTA_PC                = 00000003
ASHP_0                            00000227 R      02
ASHP_20                           0000021C R      02
ASHP_8                            00000224 R      02
ASHP_ACCVIO                       000001DA R      02
ASHP_A_DSTADDR                  = 0000000C
ASHP_BSBW_20                      00000219 R      02
ASHP_BSBW_8                       00000221 R      02
ASHP_B_CNT                      = 00000002
ASHP_B_DELTA_PC                 = 00000003
ASHP_B_ROUND                    = 0000000A
ASHP_COPY_SOURCE                  000001B7 R      02
ASHP_SHIFT_MASK                 = F0F0F0F0
ASHP_SHIFT_NEGATIVE               000000CB R      02
ASHP_SHIFT_POSITIVE               000000AA R      02
ASHP_S_ROUND                    = 00000004
ASHP_V_ROUND                    = 00000000
ASHP_W_DSTLEN                   = 00000008
CVTPS_B_DELTA_PC                = 00000003
CVTPT_B_DELTA_PC                = 00000003
CVTSP_B_DELTA_PC                = 00000003
CVTTP_B_DELTA_PC                = 00000003
DECIMAL$BOUNDS_CHECK              00000207 RG     02
DECIMAL$STRIP_ZEROS_R0_R1         ********    X   00
DECIMAL_ROPRAND                   000001D1 R      02
DIVP_B_DELTA_PC                 = 00000003
EXCEPTION_PSL                   = 0000002C
HANDLER_TABLE_BASE                00000000 R      04
MODULE_BASE                     = 00000000 R      02
MODULE_END                      = 00000244 R      02
MULP_B_DELTA_PC                 = 00000003
PACK_L_SAVED_R0                 = 00000000
PACK_L_SAVED_R2                 = 00000008
PACK_M_ACCVIO                   = 00000200
PACK_M_FPD                      = 00000100
PC_TABLE_BASE                     00000000 R      03
PSL$M_C                         = 00000001
PSL$M_N                         = 00000008
PSL$M_V                         = 00000002
PSL$M_Z                         = 00000004
PSL$V_DV                        = 00000007
PSL$V_N                         = 00000003
PSL$V_V                         = 00000001
SRM$K_DEC_OVF_T                 = 00000006
SUBP4_B_DELTA_PC                = 00000003
SUBP6_B_DELTA_PC                = 00000003
TABLE_SIZE                      = 0000000E
VAX$ADD_PACKED_BYTE_R6_R7         ********    X   00
VAX$ASHP                          00000000 RG     02
VAX$DECIMAL_ACCVIO                00000227 RG     02
VAX$DECIMAL_EXIT                  00000182 RG     02
VAX$DECIMAL_OVERFLOW              000001B0 RG     02
VAX$EDITPC_OVERFLOW               00000194 RG     02
VAX$EXIT_EMULATOR                 ********    X   00
VAX$REFLECT_FAULT                 ********    X   00
VAX$REFLECT_TRAP                  ********    X   00
VAX$ROPRAND                       ********    X   00
```

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

PSECT name                  Allocation           PSECT No.  Attributes
----------                  ----------           ---------  ----------
.  ABS  .                   00000000  (    0.)   00 (  0.)  NOPIC   USR   CON   ABS   LCL NOSHR NOEXE NORD   NOWRT NOVEC BYTE
$ABS$                       00000000  (    0.)   01 (  1.)  NOPIC   USR   CON   ABS   LCL NOSHR   EXE   RD     WRT NOVEC BYTE
_VAX$CODE                   00000244  (  580.)   02 (  2.)    PIC   USR   CON   REL   LCL   SHR   EXE   RD   NOWRT NOVEC LONG
PC_TABLE                    0000001C  (   28.)   03 (  3.)    PIC   USR   CON   REL   LCL   SHR NOEXE   RD   NOWRT NOVEC BYTE
HANDLER_TABLE               0000001C  (   28.)   04 (  4.)    PIC   USR   CON   REL   LCL   SHR NOEXE   RD   NOWRT NOVEC BYTE

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

Phase                    Page faults    CPU Time       Elapsed Time
-----                    -----------    --------       ------------
Initialization                  16    00:00:00.03      00:00:02.51
Command processing              74    00:00:00.47      00:00:05.74
Pass 1                         155    00:00:04.64      00:00:24.66
Symbol table sort                0    00:00:00.33      00:00:01.46
Pass 2                         204    00:00:02.24      00:00:11.10
Symbol table output              7    00:00:00.07      00:00:00.30
Psect syncpsis output            3    00:00:00.03      00:00:00.03
Cross-reference output           0    00:00:00.00      00:00:00.00
Assembler run totals           459    00:00:07.81      00:00:45.80

The working set limit was 1200 pages.
26813 bytes (53 pages) of virtual memory were used to buffer the intermediate code.
There were 20 pages of symbol table space allocated to hold 258 non-local and 37 local symbols.
1145 source lines were read in Pass 1, producing 19 object records in Pass 2.
30 pages of virtual memory were used to define 28 macros.

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

Macro library name                        Macros defined
------------------                        --------------
_$255$DUA28:[EMULAT.OBJ]VAXMACROS.MLB;1               19
_$255$DUA28:[SYSLIB]STARLET.MLB;2                      6
TOTALS (all libraries)                                25

429 GETS were required to define 25 macros.

There were no errors, warnings or information messages.

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

0143 AH-BT13A-SE
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY

FPLOAD
LIS

VAXASHP
LIS

VAXCONVRT
LIS

VAXARITH
LIS