```
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
```

FILEID**VAXARITH

M 8

```
VV      VV    AAAAAA    XX      XX    AAAAAA    RRRRRRRR      IIIIII    TTTTTTTTTT  HH      HH
VV      VV    AAAAAA    XX      XX    AAAAAA    RRRRRRRR      IIIIII    TTTTTTTTTT  HH      HH
VV      VV  AA      AA  XX      XX  AA      AA  RR      RR      II          TT      HH      HH
VV      VV  AA      AA  XX      XX  AA      AA  RR      RR      II          TT      HH      HH
VV      VV  AA      AA    XX  XX    AA      AA  RR      RR      II          TT      HH      HH
VV      VV  AA      AA    XX  XX    AA      AA  RR      RR      II          TT      HH      HH
VV      VV  AA      AA      XX      AA      AA  RRRRRRRR        II          TT      HHHHHHHHHH
VV      VV  AA      AA      XX      AA      AA  RRRRRRRR        II          TT      HHHHHHHHHH
VV      VV  AAAAAAAAAA    XX  XX    AAAAAAAAAA  RR  RR          II          TT      HH      HH
VV      VV  AAAAAAAAAA    XX  XX    AAAAAAAAAA  RR  RR          II          TT      HH      HH
  VV  VV    AA      AA  XX      XX  AA      AA  RR    RR        II          TT      HH      HH      ....
  VV  VV    AA      AA  XX      XX  AA      AA  RR    RR        II          TT      HH      HH      ....
    VV      AA      AA  XX      XX  AA      AA  RR      RR    IIIIII        TT      HH      HH      ....
    VV      AA      AA  XX      XX  AA      AA  RR      RR    IIIIII        TT      HH      HH      ....

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$DECIMAL_ARITHMETIC - VAX-11 Packed Decimal Arithmetic Instructio
0000          .IDENT  /V04-000/
0000
0000  ;
0000  ;*****************************************************************************
0000  ;*                                                                           *
0000  ;*  COPYRIGHT (c) 1978, 1980, 1982, 1984 BY                                  *
0000  ;*  DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.                   *
0000  ;*  ALL RIGHTS RESERVED.                                                     *
0000  ;*                                                                           *
0000  ;*  THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED    *
0000  ;*  ONLY IN  ACCORDANCE WITH  THE  TERMS  OF  SUCH  LICENSE  AND WITH THE    *
0000  ;*  INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR  ANY  OTHER    *
0000  ;*  COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY    *
0000  ;*  OTHER PERSON.  NO TITLE TO AND OWNERSHIP OF  THE  SOFTWARE IS  HEREBY    *
0000  ;*  TRANSFERRED.                                                             *
0000  ;*                                                                           *
0000  ;*  THE INFORMATION IN THIS SOFTWARE IS  SUBJECT TO CHANGE WITHOUT NOTICE    *
0000  ;*  AND  SHOULD  NOT  BE  CONSTRUED AS  A COMMITMENT BY DIGITAL EQUIPMENT    *
0000  ;*  CORPORATION.                                                             *
0000  ;*                                                                           *
0000  ;*  DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE  OR  RELIABILITY OF ITS    *
0000  ;*  SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.                  *
0000  ;*                                                                           *
0000  ;*                                                                           *
0000  ;*****************************************************************************
0000  ;
0000
0000  ;++
0000  ; Facility:
0000  ;
0000  ;       VAX-11 Instruction Emulator
0000  ;
0000  ; Abstract:
0000  ;
0000  ;       The routines in this module emulate the VAX-11 packed decimal
0000  ;       instructions that perform arithmetic operations. These procedures can
0000  ;       be a part of an emulator package or can be called directly after the
0000  ;       input parameters have been loaded into the architectural registers.
0000  ;
0000  ;       The input parameters to these routines are the registers that
0000  ;       contain the intermediate instruction state.
0000  ;
0000  ; Environment:
0000  ;
0000  ;       These routines run at any access mode, at any IPL, and are AST
0000  ;       reentrant.
0000  ;
0000  ; Author:
0000  ;
0000  ;       Lawrence J. Kenah
0000  ;
0000  ; Creation Date
0000  ;
0000  ;       19 October 1983
0000  ;
0000  ; Modified by:
```

```
0000   ;
0000   ;       V01-003 LJK0037         Lawrence J. Kenah       17-Jul-1984
0000   ;               Fix two minor bugs in exception handling code that caused
0000   ;               MULP and DIVP tests to generate spurious access violations.
0000   ;
0000   ;       V01-002 LJK0024         Lawrence J. Kenah       21-Feb-1984
0000   ;               Add code to handle access violations. Perform minor cleanup.
0000   ;               Eliminate double use of R10 in MULP and DIVP.
0000   ;
0000   ;       V01-001 LJK0008         Lawrence J. Kenah       19-Oct-1983
0000   ;               The emulation code for ADDP4, ADDP6, SUBP4, SUBP6, MULP and
0000   ;               DIVP 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          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          $PSLDEF                         ; Define bit fields in PSL
          0000          $SRMDEF                         ; Define arithmetic trap codes
          0000
          0000          .DISABLE        SUPPRESSION     ; Turn on symbol table again
          0000          .CROSS                          ; Cross reference is OK now
          0000
          0000  ; Symbol definitions
          0000
          0000  ;       The architecture requires that R4 be zero on completion of an ADDP6 or
          0000  ;       SUBP6 instruction. If we did not have to worry about restarting
          0000  ;       instructions after an access violation, we could simply zero the saved
          0000  ;       R4 value on the code path that these two instructions have in common
          0000  ;       before they merge with the ADDP4 and SUBP4 routines. The ability to
          0000  ;       restart requires that we keep the original R4 around at least until no
          0000  ;       more access violations are possible. To accomplish this, we store the
          0000  ;       fact that R4 must be cleared on exit in R11, which also contains the
          0000  ;       evolving condition codes. We use bit 31, the compatibility mode bit
          0000  ;       because it is nearly impossible to enter the emulator with CM set.
          0000
0000001F  0000          ADD_SUB_V_ZERO_R4 = PSL$V_CM
          0000
          0000  ; External declarations
          0000
          0000          .DISABLE        GLOBAL
          0000
          0000          .EXTERNAL -
          0000                          DECIMAL$BOUNDS_CHECK,-
          0000                          DECIMAL$BINARY_TO_PACKED_TABLE,-
          0000                          DECIMAL$PACKED_TO_BINARY_TABLE,-
          0000                          DECIMAL$STRIP_ZEROS_R0_R1,-
          0000                          DECIMAL$STRIP_ZEROS_R2_R3
          0000
          0000          .EXTERNAL -
          0000                          VAX$DECIMAL_EXIT,-
          0000                          VAX$DECIMAL_ACCVIO,-
          0000                          VAX$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   129
0000   130         BEGIN_MARK_POINT
```

```
                  0000
                  0000          .SUBTITLE       VAX$SUBP6 - Subtract Packed (6 Operand Format)
                  0000   ;+
                  0000   ; Functional Description:
                  0000   ;
                  0000   ;      In 6 operand format, the subtrahend string specified by  the  subtrahend
                  0000   ;      length  and  subtrahend  address operands is subtracted from the minuend
                  0000   ;      string specified by the minuend length  and  minuend  address  operands.
                  0000   ;      The  difference string specified by the difference length and difference
                  0000   ;      address operands is replaced by the result.
                  0000   ;
                  0000   ; Input Parameters:
                  0000   ;
                  0000   ;      R0 - sublen.rw          Number of digits in subtrahend string
                  0000   ;      R1 - subaddr.ab         Address of subtrahend string
                  0000   ;      R2 - minlen.rw          Number of digits in minuend string
                  0000   ;      R3 - minaddr.ab         Address of minuend string
                  0000   ;      R4 - diflen.rw          Number of digits in difference string
                  0000   ;      R5 - difaddr.ab         Address of difference string
                  0000   ;
                  0000   ; Output Parameters:
                  0000   ;
                  0000   ;      R0 = 0
                  0000   ;      R1 = Address of the byte containing the most significant digit of
                  0000   ;           the subtrahend string
                  0000   ;      R2 = 0
                  0000   ;      R3 = Address of the byte containing the most significant digit of
                  0000   ;           the minuend string
                  0000   ;      R4 = 0
                  0000   ;      R5 = Address of the byte containing the most significant digit of
                  0000   ;           the string containing the difference
                  0000   ;
                  0000   ; Condition Codes:
                  0000   ;
                  0000   ;      N <- difference string LSS 0
                  0000   ;      Z <- difference string EQL 0
                  0000   ;      V <- decimal overflow
                  0000   ;      C <- 0
                  0000   ;
                  0000   ; Register Usage:
                  0000   ;
                  0000   ;      This routine uses all of the general registers. The condition codes
                  0000   ;      are recorded in R11 as the routine executes.
                  0000   ;-
                  0000
                  0000          .ENABLE         LOCAL_BLOCK
                  0000
                  0000   VAX$SUBP6::
 0FFF 8F    BB    0000          PUSHR   #^M<R0,R1,R2,R3,R4,R5,R6,R7,R8,R9,R10,R11>      ; Save the lot
   59 01    9A    0004          MOVZBL  #1,R9                       ; Indicate that this is subtraction
      06    11    0007          BRB     10$                         ; Merge with ADDP6 code
```

```
                   0009        .SUBTITLE       VAX$ADDP6 - Add Packed (6 Operand Format)
                   0009   ;+
                   0009   ; Functional Description:
                   0009   ;
                   0009   ;       In 6 operand format, the addend 1  string  specified  by  the  addend  1
                   0009   ;       length  and  addend  1  address operands is added to the addend 2 string
                   0009   ;       specified by the addend 2 length and addend 2 address operands.  The sum
                   0009   ;       string  specified by the sum length and sum address operands is replaced
                   0009   ;       by the result.
                   0009   ;
                   0009   ; Input Parameters:
                   0009   ;
                   0009   ;       R0 - add1len.rw         Number of digits in first addend string
                   0009   ;       R1 - add1addr.ab        Address of first addend string
                   0009   ;       R2 - add2len.rw         Number of digits in second addend string
                   0009   ;       R3 - add2addr.ab        Address of second addend string
                   0009   ;       R4 - sumlen.rw          Number of digits in sum string
                   0009   ;       R5 - sumaddr.ab         Address of sum string
                   0009   ;
                   0009   ; Output Parameters:
                   0009   ;
                   0009   ;       R0 = 0
                   0009   ;       R1 = Address of the byte containing the most significant digit of
                   0009   ;            the first addend string
                   0009   ;       R2 = 0
                   0009   ;       R3 = Address of the byte containing the most significant digit of
                   0009   ;            the second addend string
                   0009   ;       R4 = 0
                   0009   ;       R5 = Address of the byte containing the most significant digit of
                   0009   ;            the string containing the sum
                   0009   ;
                   0009   ; Condition Codes:
                   0009   ;
                   0009   ;       N <- sum string LSS 0
                   0009   ;       Z <- sum string EQL 0
                   0009   ;       V <- decimal overflow
                   0009   ;       C <- 0
                   0009   ;
                   0009   ; Register Usage:
                   0009   ;
                   0009   ;       This routine uses all of the general registers. The condition codes
                   0009   ;       are recorded in R11 as the routine executes.
                   0009   ;-
                   0009
                   0009   VAX$ADDP6::
   0FFF 8F   BB    0009        PUSHR   #^M<R0,R1,R2,R3,R4,R5,R6,R7,R8,R9,R10,R11>      ; Save the lot
        59   D4    000D        CLRL    R9                              ; This is addition
                   000F   10$: ROPRAND_CHECK   R4                      ; Insure that R4 is LEQU 31
        5B   DC    001A        MOVPSL  R11                             ; Get initial PSL
                   001C
                   001C   ; Indicate that the saved R4 must be cleared on the exit path
                   001C
 1D 5B  1F   E3    001C        BBCS    #ADD_SUB_V_ZERO_R4,R11,30$      ; Set bit and join common code
        1B   11    0020        BRB     30$                             ; In case we drop through BBCS
```

```
                    0022          .SUBTITLE       VAX$SUBP4 - Subtract Packed (4 Operand Format)
                    0022  ;+
                    0022  ; Functional Description:
                    0022  ;
                    0022  ;       In 4 operand format,  the  subtrahend  string  specified  by  subtrahend
                    0022  ;       length and subtrahend address operands is subtracted from the difference
                    0022  ;       string  specified  by  the  difference  length  and  difference  address
                    0022  ;       operands and the difference string is replaced by the result.
                    0022  ;
                    0022  ; Input Parameters:
                    0022  ;
                    0022  ;       R0 - sublen.rw          Number of digits in subtrahend string
                    0022  ;       R1 - subaddr.ab         Address of subtrahend decimal string
                    0022  ;       R2 - diflen.rw          Number of digits in difference string
                    0022  ;       R3 - difaddr.ab         Address of difference decimal string
                    0022  ;
                    0022  ; Output Parameters:
                    0022  ;
                    0022  ;       R0 = 0
                    0022  ;       R1 = Address of the byte containing the most significant digit of
                    0022  ;            the subtrahend string
                    0022  ;       R2 = 0
                    0022  ;       R3 = Address of the byte containing the most significant digit of
                    0022  ;            the string containing the difference
                    0022  ;
                    0022  ; Condition Codes:
                    0022  ;
                    0022  ;       N <- difference string LSS 0
                    0022  ;       Z <- difference string EQL 0
                    0022  ;       V <- decimal overflow
                    0022  ;       C <- 0
                    0022  ;
                    0022  ; Register Usage:
                    0022  ;
                    0022  ;       This routine uses all of the general registers. The condition codes
                    0022  ;       are recorded in R11 as the routine executes.
                    0022  ;-
                    0022
                    0022  VAX$SUBP4::
0FFF 8F   BB        0022          PUSHR   #^M<R0,R1,R2,R3,R4,R5,R6,R7,R8,R9,R10,R11>      ; Save the lot
59   01   9A        0026          MOVZBL  #1,R9                           ; Indicate that this is subtraction
     06   11        0029          BRB     20$                             ; Merge with ADDP4 code
```

```
                     002B        .SUBTITLE      VAX$ADDP4 - Add Packed (4 Operand Format)
                     002B  ;+
                     002B  ; Functional Description:
                     002B  ;
                     002B  ;       In 4 operand format, the addend string specified by  the  addend  length
                     002B  ;       and  addend address operands is added to the sum string specified by the
                     002B  ;       sum length and sum address operands and the sum string  is  replaced  by
                     002B  ;       the result.
                     002B  ;
                     002B  ; Input Parameters:
                     002B  ;
                     002B  ;       R0 - addlen.rw          Number of digits in addend string
                     002B  ;       R1 - addaddr.ab         Address of addend decimal string
                     002B  ;       R2 - sumlen.rw          Number of digits in sum string
                     002B  ;       R3 - sumaddr.ab         Address of sum decimal string
                     002B  ;
                     002B  ; Output Parameters:
                     002B  ;
                     002B  ;       R0 = 0
                     002B  ;       R1 = Address of the byte containing the most significant digit of
                     002B  ;            the addend string
                     002B  ;       R2 = 0
                     002B  ;       R3 = Address of the byte containing the most significant digit of
                     002B  ;            the string containing the sum
                     002B  ;
                     002B  ; Conditior Codes:
                     002B  ;
                     002B  ;       N <- sum string LSS 0
                     002B  ;       Z <- sum string EQL 0
                     002B  ;       V <- decimal overflow
                     002B  ;       C <- 0
                     002B  ;
                     002B  ; Register Usage:
                     002B  ;
                     002B  ;       This routine uses all of the general registers. The condition codes
                     002B  ;       are recorded in R11 as the routine executes.
                     002B  ;-
                     002B
                     002B  VAX$ADDP4::
      0FFF 8F    BB  002B        PUSHR   #^M<R0,R1,R2,R3,R4,R5,R6,R7,R8,R9,R10,R11>      ; Save the lot
           59    D4  002F        CLRL    R9                              ; This is addition
                     0031
                     0031  ; The output string, described by R4 and R5, will be the same as the input
                     0031  ; string for ADDP4 and SUBP4. It is necessary to explicitly clear R4<31:16>
                     0031  ; along this code path so MOVQ R2,R4 will not always work.
                     0031
      54   52    3C  0031  20$:  MOVZWL  R2,R4                           ; Set output size equal to input size
      55   53    D0  0034        MOVL    R3,R5                           ; ... and ditto for string addresses
           5B    DC  0037        MOVPSL  R11                             ; Get initial PSL
                     0039
                     0039  ; Indicate that the saved R4 will be restored on the common exit path
                     0039
   00 5B   1F    E5  0039        BBCC    #ADD_SUB_V_ZERO_R4,R11,30$      ; Clear bit and join common code
```

```
                            003D          .SUBTITLE       ADDPx/SUBPx Common Initialization Code
                            003D  ;+
                            003D  ; All four routines converge at this point and execute common initialization
                            003D  ; code until a later decision is made to do addition or subtraction.
                            003D  ;
                            003D  ;       R4 - Number of digits in destination string
                            003D  ;       R5 - Address of destination string
                            003D  ;
                            003D  ;       R9 - Indicates whether operation is addition or subtraction
                            003D  ;               0 => addition
                            003D  ;               1 => subtraction
                            003D  ;
                            003D  ;       R11<31> - Indicates whether this is a 4-operand or 6-operand instruction
                            003D  ;               0 => 4-operand (restore saved R4 on exit)
                            003D  ;               1 => 6-operand (set R4 to zero on exit)
                            003D  ;-
                            003D
 5B   04   00   04   F0     003D  30$:    INSV    #PSL$M_Z,#0,#4,R11      ; Set Z-bit, clear the rest in saved PSW
                            0042          ESTABLISH_HANDLER       -       ; Store address of access
                            0042                  ARITH_ACCVIO            ;  violation handler
                            0047
                            0047          ROPRAND_CHECK   R2                      ; Insure that R2 is LEQU 31
                            004F          MARK_POINT      ADD_SUB_BSBW_0
           FFAE'   30       004F          BSBW    DECIMAL$STRIP_ZEROS_R2_R3       ; Strip high order zeros from R2/R3
                            0052
                            0052          ROPRAND_CHECK   R0                      ; Insure that R0 is LEQU 31
                            005A          MARK_POINT      ADD_SUB_BSBW_0
           FFA3'   30       005A          BSBW    DECIMAL$STRIP_ZEROS_R0_R1       ; Strip high order zeros from R0/R1
                            005D
                            005D  ; Rather than totally confuse the already complicated logic dealing with
                            005D  ; different length strings in the add or subtract loop, we will put the
                            005D  ; result into an intermediate buffer on the stack. This buffer will be long
                            005D  ; enough to handle the worst case so that the addition loop need only concern
                            005D  ; itself with the lengths of the two input loops. The required length is 17
                            005D  ; bytes, to handle an addition with a carry out of the most significant byte.
                            005D  ; We will allocate 20 bytes to maintain whatever alignment the stack has.
                            005D
              7E   7C       005D          CLRQ    -(SP)                   ; Set aside space for output string
              7E   7C       005F          CLRQ    -(SP)                   ; Worst case string needs 16 bytes
              7E   D4       0061          CLRL    -(SP)                   ; Add slack for a CARRY
 58   54   04   01   EF     0063          EXTZV   #1,#4,R4,R8             ; Get byte count of destination string
      7E   55   58   C1     0068          ADDL3   R8,R5,-(SP)             ; Save high address end of destination
        55   18 AE   9E     006C          MOVAB   24(SP),R5               ; Point R5 one byte beyond buffer
                            0070
                            0070  ; The number of minus signs will determine whether the real operation that we
                            0070  ; perform is addition or subtraction. That is, two plus signs or two minus
                            0070  ; signs will both result in addition, while a plus sign and a minus sign will
                            0070  ; result in subtraction. The addition and subtraction routines have their own
                            0070  ; methods for determining the correct sign of the result.
                            0070  ;
                            0070  ; For the purpose of counting minus signs, we treat subtraction as the
                            0070  ; addition of the negative of the input operand. That is, subtraction of a
                            0070  ; positive quantity causes the sign to be remembered as minus and counted as
                            0070  ; a minus sign while subtraction of a minus quantity stores a plus sign and
                            0070  ; counts nothing.
                            0070  ;
                            0070  ; On input to this code sequence, R9 distinguished addition from subtraction.
```

```
                            0070  ; On output, it contains either 0, 1, or 2, indicating the total number of
                            0070  ; minus signs, real or implied, that we counted.
                            0070
   56   50   04   01   EF   0070          EXTZV   #1,#4,R0,R6                 ; Get byte count for first input string
             51   56   C0   0075          ADDL    R6,R1                       ; Point R1 to byte containing sign
                            0078          MARK_POINT      ADD_SUB_24
    56   61   F0 8F   8B    0078          BICB3   #^B11110000,(R1),R6         ; R6 contains the sign "digit"
              10 59   E8    007D          BLBS    R9,35$                      ; Use second CASE if subtraction
                            0080
                            0080  ; This case statement is used for addition
                            0080
                            0080          CASE    R6,TYPE=B,LIMIT=#10,<-      ; Dispatch on sign digit
                            0080                  50$,-                       ; 10 => sign is "+"
                            0080                  40$,-                       ; 11 => sign is "-"
                            0080                  50$,-                       ; 12 => sign is "+"
                            0080                  40$,-                       ; 13 => sign is "-"
                            0080                  50$,-                       ; 14 => sign is "+"
                            0080                  50$,-                       ; 15 => sign is "+"
                            0080                  >
                            0090
                            0090  ; This case statement is used for subtraction
                            0090
                            0090  35$:    CASE    R6,TYPE=B,LIMIT=#10,<-      ; Dispatch on sign digit
                            0090                  40$,-                       ; 10 => treat sign as "-"
                            0090                  50$,-                       ; 11 => treat sign as "+"
                            0090                  40$,-                       ; 12 => treat sign as "-"
                            0090                  50$,-                       ; 13 => treat sign as "+"
                            0090                  40$,-                       ; 14 => treat sign as "-"
                            0090                  40$,-                       ; 15 => treat sign as "-"
                            0090                  >
                            00A0
             59   01   D0   00A0  40$:    MOVL    #1,R9                       ; Count a minus sign
             56   0D   9A   00A3          MOVZBL  #13,R6                      ; The preferred minus sign is 13
                  05   11   00A6          BRB     60$                         ; Now check second input sign
                            00A8
                  59   D4   00A8  50$:    CLRL    R9                          ; No real minus signs so far
             56   0C   9A   00AA          MOVZBL  #12,R6                      ; The preferred minus sign is 12
                            00AD
   57   52   04   01   EF   00AD  60$:    EXTZV   #1,#4,R2,R7                 ; Get byte count for second input string
             53   57   C0   00B2          ADDL    R7,R3                       ; Point R3 to byte containing sign
                            00B5          MARK_POINT      ADD_SUB_24
    57   63   F0 8F   8B    00B5          BICB3   #^B11110000,(R3),R7         ; R7 contains the sign "digit"
                            00BA
                            00BA          CASE    R7,TYPE=B,LIMIT=#10,<-      ; Dispatch on sign digit
                            00BA                  80$,-                       ; 10 => sign is "+"
                            00BA                  70$,-                       ; 11 => sign is "-"
                            00BA                  80$,-                       ; 12 => sign is "+"
                            00BA                  70$,-                       ; 13 => sign is "-"
                            00BA                  80$,-                       ; 14 => sign is "+"
                            00BA                  80$,-                       ; 15 => sign is "+"
                            00BA                  >
                            00CA
                  59   D6   00CA  70$:    INCL    R9                          ; Remember that sign was minus
             57   0D   9A   00CC          MOVZBL  #13,R7                      ; The preferred minus sign is 13
                  03   11   00CF          BRB     90$                         ; Now check second input sign
                            00D1
             57   0C   9A   00D1  80$:    MOVZBL  #12,R7                      ; The preferred minus sign is 12
```

```
                 00D4   450
   03 59   E9    00D4   451 90$:   BLBC    R9,ADD_PACKED           ; Even parity indicates addition
                 00D7   452
    00B3   31    00D7   453        BRW     SUBTRACT_PACKED         ; Odd parity calls for subtraction
                 00DA   454
                 00DA   455        .DISABLE        LOCAL_BLOCK
```

```
00DA          .SUBTITLE       ADD_PACKED - Add Two Packed Decimal Strings
00DA   ;+
00DA   ; Functional Description:
00DA   ;
00DA   ;       This routine adds two packed decimal strings whose descriptors
00DA   ;       are passed as input parameters and places their sum into another
00DA   ;       (perhaps identical) packed decimal string.
00DA   ;
00DA   ;       At the present time, the result is placed into a 16-byte storage
00DA   ;       area while the sum is being evaluated. This drastically reduces
00DA   ;       the number of different cases that must be dealt with as each
00DA   ;       pair of bytes in the two input strings is added.
00DA   ;
00DA   ;       The signs of the two input strings have already been dealt with
00DA   ;       so this routine performs addition in all cases, even if the original
00DA   ;       entry was at SUBP4 or SUBP6. The cases that arrive in this routine
00DA   ;       are as follows.
00DA   ;
00DA   ;                            R2/R3           R0/R1           result
00DA   ;                      +---------------+---------------+---------------+
00DA   ;                      :               :               :               :
00DA   ;      R2/R3 + R0/R1   :     plus      :     plus      :     plus      :
00DA   ;                      :               :               :               :
00DA   ;                      +---------------+---------------+---------------+
00DA   ;                      :               :               :               :
00DA   ;      R2/R3 + R0/R1   :     minus     :     minus     :     minus     :
00DA   ;                      :               :               :               :
00DA   ;                      +---------------+---------------+---------------+
00DA   ;                      :               :               :               :
00DA   ;      R2/R3 - R0/R1   :     minus     :     plus      :     minus     :
00DA   ;                      :               :               :               :
00DA   ;                      +---------------+---------------+---------------+
00DA   ;                      :               :               :               :
00DA   ;      R2/R3 - R0/R1   :     plus      :     minus     :     plus      :
00DA   ;                      :               :               :               :
00DA   ;                      +---------------+---------------+---------------+
00DA   ;
00DA   ;       Note that the correct choice of sign in all four cases is the sign
00DA   ;       of the second input string, the one described by R2 and R3.
00DA   ;
00DA   ; Input Parameters:
00DA   ;
00DA   ;       R0<4:0> - Number of digits in first input decimal string
00DA   ;       R1      - Address of least significant digit of first input
00DA   ;                 decimal string (the byte containing the sign)
00DA   ;
00DA   ;       R2<4:0> - Number of digits in second input decimal string
00DA   ;       R3      - Address of least significant digit of second input
00DA   ;                 decimal string (the byte containing the sign)
00DA   ;
00DA   ;       R4<4:0> - Number of digits in output decimal string
00DA   ;       R5      - Address of one byte beyond least significant digit of
00DA   ;                 intermediate string stored on the stack
00DA   ;
00DA   ;       R6<3:0> - Sign of first input string in preferred form
00DA   ;       R7<3:0> - Sign of second input string in preferred form
00DA   ;
```

```
                        00DA      ;       R11     - Saved PSL (Z-bit is set, other condition codes are clear)
                        00DA      ;
                        00DA      ;       (SP)    - Saved R5, address of least significant digit of ultimate
                        00DA      ;                 destination string.
                        00DA      ;       4(SP)   - Beginning of 20-byte buffer to hold intermediate result
                        00DA      ;
                        00DA      ; Output Parameters:
                        00DA      ;
                        00DA      ;       The particular input operation (ADDPx or SUBPx) is completed in
                        00DA      ;       this routine. See the routine headers for the four routines that
                        00DA      ;       request addition or subtraction for a list of output parameters
                        00DA      ;       from this routine.
                        00DA      ;-
                        00DA
                        00DA      ADD_PACKED:
          59   57   90  00DA              MOVB    R7,R9                   ; Use sign of second string for output
           03 59    E9  00DD              BLBC    R9,10$                  ; Check if sign is negative
          5B   08   88  00E0              BISB    #PSL$M_N,R11            ; ... so the saved N-bit can be set
                        00E3
                        00E3              MARK_POINT      ADD_SUB_24
     56   61   0F   8B  00E3      10$:    BICB3   #^B00001111,(R1),R6     ; Get least significant digit to R6
                        00E7              MARK_POINT      ADD_SUB_24
     57   63   0F   8B  00E7              BICB3   #^B00001111,(R3),R7     ; Get least significant digit to R7
               58   D4  00EB              CLRL    R8                      ; Start the add with CARRY off
             0075   30  00ED              BSBW    ADD_PACKED_BYTE_R6_R7   ; Add the two low order digits
                        00F0
                        00F0      ; The following set of instructions computes the number of bytes in the two
                        00F0      ; strings and, if necessary, performs a switch so that R0 and R1 always
                        00F0      ; describe the shorter of the two strings.
                        00F0
 50   50   04   01   EF 00F0              EXTZV   #1,#4,R0,R0             ; Convert digit count to byte count
 52   52   04   01   EF 00F5              EXTZV   #1,#4,R2,R2             ; Do it for both strings
          52   50   D1  00FA              CMPL    R0,R2                   ; We want to compare the byte counts
               09   1B  00FD              BLEQU   20$                     ; Skip the swap if we're already correct
          56   50   7D  00FF              MOVQ    R0,R6                   ; Save the longer
          50   52   7D  0102              MOVQ    R2,R0                   ; Store the shorter on R0 and R1
          52   56   7D  0105              MOVQ    R6,R2                   ; ... and store the longer in R2 and R3
          52   50   C2  0108      20$:    SUBL    R0,R2                   ; Make R2 a difference (R2 GEQU 0)
                        010B
                        010B      ; R0 now contains the number of bytes remaining in the shorter string.
                        010B      ; R2 contains the difference in bytes between the two input strings.
                        010B
               50   D5  010B              TSTL    R0                      ; Does shorter string have any room?
               06   13  010D              BEQL    40$                     ; Skip loop if no room at all
                        010F
             004D   30  010F      30$:    BSBW    ADD_PACKED_BYTE_STRING  ; Add the next two bytes together
            FA 50   F5  0112              SOBGTR  R0,30$                  ; Check for end of loop
                        0115
               52   D5  0115      40$:    TSTL    R2                      ; Does longer string have any room?
               16   13  0117              BEQL    70$                     ; Skip next loops if all done
                        0119
            0D 58   E9  0119      50$:    BLBC    R8,60$                  ; Life is simple if CARRY clear
                        011C
               56   D4  011C              CLRL    R6                      ; Otherwise, CARRY must propogate
                        011E              MARK_POINT      ADD_SUB_24
          57   73   9A  011E              MOVZBL  -(R3),R7                ; So add CARRY to single string
             0041   30  0121              BSBW    ADD_PACKED_BYTE_R6_R7   ; Use the special entry point
```

```
            F2 52   F5 0124          SOBGTR  R2,50$                  ; Check for this string exhausted
               05   11 0127          BRB     70$                     ; Join common completion code
                       0129
                       0129          MARK_POINT      ADD_SUB_24
            75  73  90 0129  60$:    MOVB    -(R3),-(R5)             ; Simply move src to dst if no CARRY
            FA 52   F5 012C          SOBGTR  R2,60$                  ; ... until we're all done
                       012F
            75  58  90 012F  70$:    MOVB    R8,-(R5)                ; Store the final CARRY
                       0132
                       0132  ;+
                       0132  ; At this point, the result has been computed. That result must be moved to
                       0132  ; its ultimate destination, noting whether any nonzero digits are stored
                       0132  ; so that the Z-bit will have its correct setting.
                       0132  ;
                       0132  ; Input Parameters:
                       0132  ;
                       0132  ;       R9<7:0>  - Sign of result in preferred form
                       0132  ;       R11<3:0> - Saved condition codes
                       0132  ;       R11<31>  - Indicates whether to set saved R4 to zero
                       0132  ;
                       0132  ;       (SP)     - Saved R5, high address end of destination string
                       0132  ;-
                       0132
                       0132  ADD_SUBTRACT_EXIT:
        55  6E  01  C1 0132          ADDL3   #1,(SP),R5              ; Point R5 beyond real destination
         51  18 AE  9E 0136          MOVAB   24(SP),R1               ; R1 locates the saved result
              010C  30 013A          BSBW    STORE_RESULT            ; Store the result and record the Z-bit
        12 5B   02  E0 013D          BBS     #PSL$V_Z,R11,100$       ; Step out of line for minus zero check
                       0141
                       0141          MARK_POINT      ADD_SUB_24
  9E   04   00  59  F0 0141  80$:    INSV    R9,#0,#4,@(SP)+         ; The sign can finally be stored
            5E  14  C0 0146          ADDL    #20,SP                  ; Get rid of intermediate buffer
        03 5B   1F  E1 0149          BBC     #ADD_SUB_V_ZERO_R4,R11,90$      ; Branch if 4-operand opcode
            10 AE   D4 014D          CLRL    16(SP)                  ; Clear saved R4 to return zero
             FEAD'  31 0150  90$:    BRW     VAX$DECIMAL_EXIT        ; Exit through common code path
                       0153
                       0153  ; If the result is negative zero, then the N-bit is cleared and the sign
                       0153  ; is changed to a plus sign.
                       0153
            5B  08  8A 0153  100$:   BICB    #PSL$M_N,R11            ; Clear the N-bit unconditionally
        E7 5B   01  E0 0156          BBS     #PSL$V_V,R11,80$        ; Do not change the sign on overflow
            59  0C  90 015A          MOVB    #12,R9                  ; Make sure that the sign is plus
                E2  11 015D          BRB     80$                     ; ... and rejoin the exit code
```

```
015F         .SUBTITLE       ADD_PACKED_BYTE - Add Two Bytes Containing Decimal Digits
015F  ;+
015F  ; Functional Description:
015F  ;
015F  ;       This routine adds together two bytes containing decimal digits and
015F  ;       produces a byte containing the sum that is stored in the output
015F  ;       string. Each of the input bytes is converted to a binary number
015F  ;       (with a table-driven conversion), the two numbers are added, and
015F  ;       the sum is converted back to two decimal digits stored in a byte.
015F  ;
015F  ;       This routine makes no provisions for bytes that contain illegal
015F  ;       decimal digits. We are using the UNPREDICTABLE statement in the
015F  ;       architectural description of the decimal instructions to its fullest.
015F  ;
015F  ;       The bytes that contain a pair of packed decimal digits can either
015F  ;       exist in packed decimal strings located by R1 and R3 or they can
015F  ;       be stored directly in registers. In the former case, the digits must
015F  ;       be extracted from registers before they can be used in later operations
015F  ;       because the sum will be used as an index register.
015F  ;
015F  ; For entry at ADD_PACKED_BYTE_STRING:
015F  ;
015F  ;       Input Parameters:
015F  ;
015F  ;               R1  - Address one byte beyond first byte that is to be added
015F  ;               R3  - Address one byte beyond second byte that is to be added
015F  ;               R5  - Address one byte beyond location to store sum
015F  ;
015F  ;               R8  - Carry from previous byte (R8 is either 0 or 1)
015F  ;
015F  ;       Implicit Input:
015F  ;
015F  ;               R6 - Scratch
015F  ;               R7 - Scratch
015F  ;
015F  ;       Output Parameters:
015F  ;
015F  ;               R1 - Decreased by one to point to current byte in first input string
015F  ;               R3 - Decreased by one to point to current byte in second input strin
015F  ;               R5 - Decreased by one to point to current byte in output string
015F  ;
015F  ;               R8 - Either 0 or 1, reflecting whether this most recent ADD resulted
015F  ;                    in a CARRY to the next byte.
015F  ;
015F  ; For entry at ADD_PACKED_BYTE_R6_R7:
015F  ;
015F  ;       Input Parameters:
015F  ;
015F  ;               R6  - First byte containing decimal digit pair
015F  ;               R7  - Second byte containing decimal digit pair
015F  ;
015F  ;               R5  - Address one byte beyond location to store sum
015F  ;
015F  ;               R8  - Carry from previous byte (R8 is either 0 or 1)
015F  ;
015F  ;       Output Parameters:
015F  ;
```

```
                          015F   ;               R5 - Decreased by one to point to current byte in output string
                          015F   ;
                          015F   ;               R8 - Either 0 or 1, reflecting whether this most recent ADD resulted
                          015F   ;                    in a CARRY to the next byte.
                          015F   ;
                          015F   ; Side Effects:
                          015F   ;
                          015F   ;       R6 and R7 are modified by this routine
                          015F   ;
                          015F   ;       R0, R2, R4, and R9 (and, of course, R10 and R11) are preserved
                          015F   ;       by this routine
                          015F   ;
                          015F   ; Assumptions:
                          015F   ;
                          015F   ;       This routine makes two important assumptions.
                          015F   ;
                          015F   ;       1.  If both of the input bytes contain only legal decimal digits, then
                          015F   ;           it is only necessary to subtract 100 at most once to put all
                          015F   ;           possible sums in the range 0..99. That is,
                          015F   ;
                          015F   ;               99 + 99 + 1 = 199 LSS 200
                          015F   ;
                          015F   ;       2.  The result will be checked in some way to determine whether the
                          015F   ;           result is nonzero so that the Z-bit can have its correct setting.
                          015F   ;-
                          015F
                          015F   ADD_PACKED_BYTE_STRING:
                          015F
                          015F           MARK_POINT      ADD_SUB_BSBW_24
         56    71    9A   015F           MOVZBL  -(R1),R6                ; Get byte from first string
                          0162           MARK_POINT      ADD_SUB_BSBW_24
         57    73    9A   0162           MOVZBL  -(R3),R7                ; Get byte from second string
                          0165
                          0165   VAX$ADD_PACKED_BYTE_R6_R7::             ; ASHP also uses this routine
                          0165   ADD_PACKED_BYTE_R6_R7:
56    0000'CF46      90   0165           MOVB    DECIMAL$PACKED_TO_BINARY_TABLE[R6],-
                          016B                   R6                      ; Convert digits to binary
57    0000'CF47      90   016B           MOVB    DECIMAL$PACKED_TO_BINARY_TABLE[R7],-
                          0171                   R7                      ; Convert digits to binary
         57    56    80   0171           ADDB    R6,R7                   ; Form their sum
         57    58    80   0174           ADDB    R8,R7                   ; Add CARRY from last step
               58    94   0177           CLRB    R8                      ; Assume no CARRY this time
      63 8F    57    91   0179           CMPB    R7,#99                  ; Check for CARRY
               07    1B   017D           BLEQU   10$                     ; Branch if within bounds
         58    01    90   017F           MOVB    #1,R8                   ; Propogate CARRY to next step
      57    64 8F    82   0182           SUBB    #100,R7                 ; Put R7 into interval 0..99
75    0000'CF47      90   0186   10$:    MOVB    DECIMAL$BINARY_TO_PACKED_TABLE[R7],-
                          018C                   -(R5)                   ; Store converted sum byte
                     05   018C           RSB
```

```
0180        .SUBTITLE       SUBTRACT_PACKED - Subtract Two Packed Decimal Strings
018D  ;+
018D  ; Functional Description:
018D  ;
018D  ;       This routine takes two packed decimal strings whose descriptors
018D  ;       are passed as input parameters, subtracts one string from the
018D  ;       other, and places their sum into another (perhaps identical)
018D  ;       packed decimal string.
018D  ;
018D  ;       At the present time, the result is placed into a 16-byte storage
018D  ;       area while the difference is being evaluated. This drastically reduces
018D  ;       the number of different cases that must be dealt with as each
018D  ;       pair of bytes in the two input strings is added.
018D  ;
018D  ;       The signs of the two input strings have already been dealt with so
018D  ;       this routine performs subtraction in all cases, even if the original
018D  ;       entry was at ADDP4 or ADDP6.
018D  ;
018D  ; Input Parameters:
018D  ;
018D  ;       R0<4:0> - Number of digits in first input decimal string
018D  ;       R1      - Address of least significant digit of first input
018D  ;                 decimal string (the byte containing the sign)
018D  ;
018D  ;       R2<4:0> - Number of digits in second input decimal string
018D  ;       R3      - Address of least significant digit of second input
018D  ;                 decimal string (the byte containing the sign)
018D  ;
018D  ;       R4<4:0> - Number of digits in output decimal string
018D  ;       R5      - Address of one byte beyond least significant digit of
018D  ;                 intermediate string stored on the stack
018D  ;
018D  ;       R6<3:0> - Sign of first input string in preferred form
018D  ;       R7<3:0> - Sign of second input string in preferred form
018D  ;
018D  ;       R11     - Saved PSL (Z-bit is set, other condition codes are clear)
018D  ;
018D  ;       (SP)    - Saved R5, address of least significant digit of ultimate
018D  ;                 destination string.
018D  ;       4(SP)   - Beginning of 20-byte buffer to hold intermediate result
018D  ;
018D  ; Output Parameters:
018D  ;
018D  ;       The particular input operation (ADDPx or SUBPx) is completed in
018D  ;       this routine. See the routine headers for the four routines that
018D  ;       request addition or subtraction for a list of output parameters
018D  ;       from this routine.
018D  ;
018D  ; Algorithm for Choice of Sign:
018D  ;
018D  ;       The choice of sign for the output string is not nearly so
018D  ;       straightforward as it is in the case of addition. One approach that is
018D  ;       often taken is to make a reasonable guess at the sign of the result.
018D  ;       If the final subtraction causes a BORROW, then the choice was incorrect.
018D  ;       The sign must be changed and the result must be replaced by its tens
018D  ;       complement.
018D  ;
```

```
                              018D  ;       This routine does not guess. Instead, it chooses the input string of
                              018D  ;       the larger absolute magnitude as the minuend for this internal
                              018D  ;       routine and chooses its sign as the sign of the result.
                              018D  ;       This algorithm is actually more efficient than the reasonable
                              018D  ;       guess method and is probably better than a guess method that is never
                              018D  ;       wrong. All complete bytes that are processed in the sign evaluation
                              018D  ;       preprocessing loop are eliminated from consideration in the
                              018D  ;       subtraction loop, which has a higher cost per byte.
                              018D  ;
                              018D  ;       The actual algorithm is as follows. (Note that both input strings have
                              018D  ;       already had leading zeros stripped so their lengths reflect
                              018D  ;       significant digits.)
                              018D  ;
                              018D  ;       1.  If the two strings have unequal lengths, then choose the sign of
                              018D  ;           the string that has the longer length.
                              018D  ;
                              018D  ;       2.  For strings of equal length, choose the sign of the string whose
                              018D  ;           most significant byte is larger in magnitude.
                              018D  ;
                              018D  ;       3.  If the most significant bytes test equal, then decrease the
                              018D  ;           lengths of each string by one byte, drop the previous most
                              018D  ;           significant bytes, and go back to step 2.
                              018D  ;
                              018D  ;       4.  If the two strings test equal, it is not necessary to do any
                              018D  ;           subtraction. The result is identically zero.
                              018D  ;
                              018D  ;       Note that the key to this routine's efficiency is that high order
                              018D  ;       bytes that test equal in this loop are dropped from consideration in
                              018D  ;       the more complicated subtraction loop.
                              018D  ;-
                              018D
                              018D  SUBTRACT_PACKED:
      50    50    04    01 EF 018D          EXTZV   #1,#4,R0,R0             ; Convert digit count to byte count
      52    52    04    01 EF 0192          EXTZV   #1,#4,R2,R2             ; Do it for both strings
                  52    50 D1 0197          CMPL    R0,R2                   ; We want to compare the byte counts
                        3C 1F 019A          BLSSU   40$                     ; R0/R1 represent the smaller string
                        2A 1A 019C          BGTRU   30$                     ; R2/R3 represent the smaller string
                              019E
                              019E  ; The two input strings have an equal number of bytes. Compare magnitudes to
                              019E  ; determine which string is really larger. If the two strings test equal, then
                              019E  ; skip the entire subtraction loop.
                              019E
            58    51    50 C3 019E          SUBL3   R0,R1,R8                ; Point R8 to low address end of R0/R1
            59    53    52 C3 01A2          SUBL3   R2,R3,R9                ; Point R9 to low address end of R2/R3
                        50 D5 01A6          TSTL    R0                      ; See if both strings have zero bytes
                        0C 13 01A8          BEQL    20$                     ; Still need to check low order digit
                              01AA
                              01AA          MARK_POINT      ADD_SUB_24
                  89    88 91 01AA  10$:    CMPB    (R8)+,(R9)+             ; Compare most significant bytes
                        29 1F 01AD          BLSSU   40$                     ; R0/R1 represent the smaller string
                        17 1A 01AF          BGTRU   30$                     ; R2/R3 represent the smaller string
                        52 D7 01B1          DECL    R2                      ; Keep R2 in step with R0
                     F4 50 F5 01B3          SOBGTR  R0,10$                  ; ... which gets decremented here
                              01B6
                              01B6  ; At this point, we have reduced both input strings to single bytes that
                              01B6  ; contain a sign "digit" and may contain a digit in the high order nibble
                              01B6  ; if the original digit counts were nonzero.
```

```
                            01B6
                            01B6          MARK_POINT      ADD_SUB_24
        58    68    0F  8B  01B6  20$:    BICB3   #^B00001111,(R8),R8     ; Look only at digit, ignoring sign
                            01BA          MARK_POINT      ADD_SUB_24
        59    69    0F  8B  01BA          BICB3   #^B00001111,(R9),R9     ; Get the digit from the other string
              59    58  91  01BE          CMPB    R8,R9                   ; Compare these digits
                    15  1F  01C1          BLSSU   40$                     ; R0/R1 represent the smaller string
                    03  1A  01C3          BGTRU   30$                     ; R2/R3 represent the smaller string
                            01C5
                            01C5  ; The two strings have identical magnitudes. Enter the end processing code
                            01C5  ; with the intermediate result unchanged (that is, zero).
                            01C5
                  FF6A  31  01C5          BRW     ADD_SUBTRACT_EXIT       ; Join the common completion code
                            01C8
                            01C8  ; The string described by R0 and R1 has the larger magnitude. Choose its sign.
                            01C8  ; Then swap the two string descriptors so that the main subtraction loops
                            01C8  ; always have R2 and R3 describing the larger string. Note that the use of
                            01C8  ; R6 and R7 as scratch leaves R7<31:8> in an UNPREDICTABLE state.
                            01C8
              59    56  90  01C8  30$:    MOVB    R6,R9                   ; Load preferred sign into R9
              56    50  7D  01CB          MOVQ    R0,R6                   ; Save the longer
              50    52  7D  01CE          MOVQ    R2,R0                   ; Store the shorter on R0 and R1
              52    56  7D  01D1          MOVQ    R6,R2                   ; ... and store the longer in R2 and R3
                    57  D4  01D4          CLRL    R7                      ; Insure that R7<31:8> is zero
                    03  11  01D6          BRB     50$                     ; Continue along common code path
                            01D8
                            01D8  ; The string described by R2 and R3 has the larger magnitude. Choose its sign.
                            01D8
              59    57  90  01D8  40$:    MOVB    R7,R9                   ; Load preferred sign into R9
                            01DB
              52    50  C2  01DB  50$:    SUBL    R0,R2                   ; Make R2 a difference (R2 GEQU 0)
                 03 59  E9  01DE          BLBC    R9,60$                  ; Check if sign is negative
              5B    08  88  01E1          BISB    #PSL$M_N,R11            ; ... so the saved N-bit can be set
                            01E4
                            01E4          MARK_POINT      ADD_SUB_24
        56    61    0F  8B  01E4  60$:    BICB3   #^B00001111,(R1),R6     ; Get least significant digit to R6
                            01E8          MARK_POINT      ADD_SUB_24
        57    63    0F  8B  01E8          BICB3   #^B00001111,(R3),R7     ; Get least significant digit to R7
                    58  D4  01EC          CLRL    R8                      ; Start subtracting with BORROW off
                  0032  30  01EE          BSBW    SUB_PACKED_BYTE_R6_R7   ; Subtract the two low order digits
                            01F1
                            01F1  ; R0 contains the number of bytes remaining in the smaller string
                            01F1  ; R2 contains the difference in bytes between the two input strings
                            01F1
                    50  D5  01F1          TSTL    R0                      ; Does smaller string have any room?
                    06  13  01F3          BEQL    80$                     ; Skip loop if no room at all
                            01F5
                  0025  30  01F5  70$:    BSBW    SUB_PACKED_BYTE_STRING  ; Subtract the next two bytes
                 FA 50  F5  01F8          SOBGTR  R0,70$                  ; Check for end of loop
                            01FB
                    52  D5  01FB  80$:    TSTL    R2                      ; Does one of the strings have more?
                    16  13  01FD          BEQL    110$                    ; Skip next loops if all done
                            01FF
                 0D 58  E9  01FF  90$:    BLBC    R8,100$                 ; Life is simple if BORROW clear
                            0202
                    56  D4  0202          CLRL    R6                      ; Otherwise, BORROW must propogate
                            0204          MARK_POINT      ADD_SUB_24
```

```
  57   73   9A  0204           MOVZBL  -(R3),R7                  ; So subtract BORROW from single string
     0019   30  0207           BSBW    SUB_PACKED_BYTE_R6_R7     ; Use the special entry point
    F2 52   F5  020A           SOBGTR  R2,90$                    ; Check for this string exhausted
       06   11  020D           BRB     110$                      ; Join common completion code
                020F
                020F           MARK_POINT      ADD_SUB_24
  75   73   90  020F     100$: MOVB    -(R3),-(R5)               ; Simply move src to dst if no BORROW
    FA 52   F5  0212           SOBGTR  R2,100$                   ; ... until we're all done
                0215
                0215     110$:
                0215
                0215     ;;; ********** BEGIN TEMP **********
                0215     ;;;
                0215     ;;; THE FOLLOWING HALT INSTRUCTION SHOULD BE REPLACED WITH THE CORRECT
                0215     ;;; ABORT CODE.
                0215     ;;;
                0215     ;;; THE HALT IS SIMILAR TO THE
                0215     ;;;
                0215     ;;;     MICROCODE CANNOT GET HERE
                0215     ;;;
                0215     ;;; ERRORS THAT OTHER IMPLEMENTATIONS USE.
                0215     ;;;
       58   D5  0215           tstl    r8                        ; If BORROW is set here, we blew it
       01   13  0217           beql    120$                      ; Branch out if OK
            00  0219           halt                              ; This will cause an OPCDEC exception
                021A     120$:
                021A     ;;;
                021A     ;;; *********** END TEMP ***********
                021A
     FF15   31  021A           BRW     ADD_SUBTRACT_EXIT         ; Join common completion code
```

```
021D          .SUBTITLE       SUB_PACKED_BYTE - Subtract Two Bytes Containing Decimal Digi
021D  ;+
021D  ; Functional Description:
021D  ;
021D  ;       This routine takes as input two bytes containing decimal digits and
021D  ;       produces a byte containing their difference. This result is stored in
021D  ;       the output string. Each of the input bytes is converted to a binary
021D  ;       number (with a table-driven conversion), the first number is
021D  ;       subtracted from the second, and the difference is converted back to
021D  ;       two decimal digits stored in a byte.
021D  ;
021D  ;       This routine makes no provisions for bytes that contain illegal
021D  ;       decimal digits. We are using the UNPREDICTABLE statement in the
021D  ;       architectural description of the decimal instructions to its fullest.
021D  ;
021D  ;       The bytes that contain a pair of packed decimal digits can either
021D  ;       exist in packed decimal strings located by R1 and R3 or they can
021D  ;       be stored directly in registers. In the former case, the digits must
021D  ;       be extracted from registers before they can be used in later operations
021D  ;       because the difference will be used as an index register.
021D  ;
021D  ; For entry at SUB_PACKED_BYTE_STRING:
021D  ;
021D  ;       Input Parameters:
021D  ;
021D  ;               R1  - Address one byte beyond byte containing subtrahend
021D  ;               R3  - Address one byte beyond byte containing minuend
021D  ;               R5  - Address one byte beyond location to store difference
021D  ;
021D  ;               R8  - BORROW from previous byte (R8 is either 0 or 1)
021D  ;
021D  ;       Implicit Input:
021D  ;
021D  ;               R6 - Scratch
021D  ;               R7 - Scratch
021D  ;
021D  ;       Output Parameters:
021D  ;
021D  ;               R1 - Decreased by one to point to current byte
021D  ;                    in subtrahend string
021D  ;               R3 - Decreased by one to point to current byte
021D  ;                    in minuend string
021D  ;               R5 - Decreased by one to point to current byte
021D  ;                    in difference string
021D  ;
021D  ;               R8 - Either 0 or 1, reflecting whether this most recent
021D  ;                    subtraction resulted in a BORROW from the next byte.
021D  ;
021D  ; For entry at SUB_PACKED_BYTE_R6_R7:
021D  ;
021D  ;       Input Parameters:
021D  ;
021D  ;               R6<7:0>  - Byte containing decimal digit pair for subtrahend
021D  ;               R6<31:8> - MBZ
021D  ;               R7<7:0>  - Byte containing decimal digit pair for minuend
021D  ;               R7<31:8> - MBZ
021D  ;
```

```
                          021D  ;               R5 - Address one byte beyond location to store difference
                          021D  ;
                          021D  ;               R8 - BORROW from subtraction of previous byte
                          021D  ;                    (R8 is either 0 or 1)
                          021D  ;
                          021D  ;       Output Parameters:
                          021D  ;
                          021D  ;               R5 - Decreased by one to point to current byte
                          021D  ;                    in difference string
                          021D  ;
                          021D  ;               R8 - Either 0 or 1, reflecting whether this most recent
                          021D  ;                    subtraction resulted in a BORROW from the next byte.
                          021D  ;
                          021D  ; Side Effects:
                          021D  ;
                          021D  ;       R6 and R7 are modified by this routine
                          021D  ;
                          021D  ;       R0, R2, R4, and R9 (and, of course, R10 and R11) are preserved
                          021D  ;       by this routine
                          021D  ;
                          021D  ; Assumptions:
                          021D  ;
                          021D  ;       This routine makes two important assumptions.
                          021D  ;
                          021D  ;       1.  If both of the input bytes contain only legal decimal digits, then
                          021D  ;           it is only necessary to add 100 at most once to put all
                          021D  ;           possible differences in the range 0..99. That is,
                          021D  ;
                          021D  ;               0 - 99 - 1 = -100
                          021D  ;
                          021D  ;       2.  The result will be checked in some way to determine whether the
                          021D  ;           result is nonzero so that the Z-bit can have its correct setting.
                          021D  ;-
                          021D
                          021D  SUB_PACKED_BYTE_STRING:
                          021D
                          021D          MARK_POINT      ADD_SUB_BSBW_24
          56    71    9A  021D          MOVZBL  -(R1),R6                ; Get byte from first string
                          0220          MARK_POINT      ADD_SUB_BSBW_24
          57    73    9A  0220          MOVZBL  -(R3),R7                ; Get byte from second string
                          0223
                          0223  SUB_PACKED_BYTE_R6_R7:
  56    0000'CF46     90  0223          MOVB    DECIMAL$PACKED_TO_BINARY_TABLE[R6],-
                          0229                          R6              ; Convert digits to binary
  57    0000'CF47     90  0229          MOVB    DECIMAL$PACKED_TO_BINARY_TABLE[R7],-
                          022F                          R7              ; Convert digits to binary
          57    56    82  022F          SUBB    R6,R7                   ; Form their difference
          57    58    82  0232          SUBB    R8,R7                   ; Include BORROW from last step
                04    19  0235          BLSS    10$                     ; Branch if need to BORROW
                58    94  0237          CLRB    R8                      ; No BORROW next time
                07    11  0239          BRB     20$                     ; Join common exit code
                          023B
       57    64 8F    80  023B  10$:    ADDB    #100,R7                 ; Put R7 into interval 0..99
          58    01    90  023F          MOVB    #1,R8                   ; Propogate BORROW to next step
                          0242
  75    0000'CF47     90  0242  20$:    MOVB    DECIMAL$BINARY_TO_PACKED_TABLE[R7],-
                          0248                          -(R5)           ; Store converted sum byte
```

```
05  0248  1038          RSB
```

```
                        0249          .SUBTITLE       STORE_RESULT - Store Decimal String
                        0249     ;+
                        0249     ; Functional Description:
                        0249     ;
                        0249     ;       This routine takes a packed decimal string that typically contains
                        0249     ;       the result of an arithmetic operation and stores it in another
                        0249     ;       decimal string whose descriptor is specified as an input parameter
                        0249     ;       to the original arithmetic operation.
                        0249     ;
                        0249     ;       The string is stored from the high address end (least significant
                        0249     ;       digits) to the low address end (most significant digits). This order
                        0249     ;       allows all of the special cases to be handled in the simplest fashion.
                        0249     ;
                        0249     ; Input Parameters:
                        0249     ;
                        0249     ;       R1      - Address one byte beyond high address end of input string
                        0249     ;                 (Note that this string must be at least 17 bytes long.)
                        0249     ;
                        0249     ;       R4<4:0> - Number of digits in ultimate destination
                        0249     ;       R5      - Address one byte beyond destination string
                        0249     ;
                        0249     ;       R11     - Contains saved condition codes
                        0249     ;
                        0249     ; Implicit Input:
                        0249     ;
                        0249     ;       The input string must be at least 17 bytes long to contain a potential
                        0249     ;       carry out of the highest digit when doing an add of two large numbers.
                        0249     ;       This carry out of the last byte will be detected and reported as a
                        0249     ;       decimal overflow, either as an exception or simply by setting the V-bit.
                        0249     ;
                        0249     ;       The least significant digit (highest addressed byte) cannot contain a
                        0249     ;       sign digit because that would cause the Z-bit to be incorrectly cleared.
                        0249     ;
                        0249     ; Output Parameters:
                        0249     ;
                        0249     ;       R11<PSL$V_Z> - Cleared if a nonzero digit is stored in output string
                        0249     ;       R11<PSL$V_V> - Set if a nonzero digit is detected after the output
                        0249     ;                      string is exhausted
                        0249     ;
                        0249     ;       A portion of the result (dictated by the size of R4 on input) is
                        0249     ;       moved to the destination string.
                        0249     ;-
                        0249
                        0249     STORE_RESULT:
             54    D6   0249          INCL    R4                      ; Want number of "complete" bytes in
 50   54  FF 8F    78   024B          ASHL    #-1,R4,R0               ;  output string
             0B    13   0250          BEQL    30$                     ; Skip first loop if none
                        0252
                        0252          MARK_POINT      ADD_SUB_BSBW_24
          75 71    90   0252     10$: MOVB    -(R1),-(R5)             ; Move the next complete byte
             03    13   0255          BEQL    20$                     ; Check whether to clear Z-bit
          5B 04    8A   0257          BICB    #PSL$M_Z,R11            ; Clear Z-bit if nonzero
          F5 50    F5   025A     20$: SOBGTR  R0,10$                  ; Keep going?
                        025D
          10 54    E9   025D     30$: BLBC    R4,50$                  ; Was original R4 odd? Branch if yes
                        0260          MARK_POINT      ADD_SUB_BSBW_24
 75   71  F0 8F    8B   0260          BICB3   #^B11110000,-(R1),-(R5) ; If R4 was even, store half a byte
```

```
                 03   13 0265         BEQL    40$                     ; Need to check for zero here, too
            5B   04   8A 0267         BICB    #PSL$M_Z,R11            ; Clear Z-bit if nonzero
                         026A         MARK_POINT      ADD_SUB_BSBW_24
     61     F0 8F     93 026A  40$:   BITB    #^B11110000,(R1)        ; If high order nibble is nonzero,
                 13   12 026E         BNEQ    70$                     ; ... then overflow has occurred
                         0270
                         0270  ; The entire destination has been stored. We must now check whether any of
                         0270  ; the remaining input string is nonzero and set the V-bit if nonzero is
                         0270  ; detected. Note that at least one byte of the output string has been examined
                         0270  ; in all cases already. This makes the next byte count calculation correct.
                         0270
                      54   D7 0270  50$:   DECL    R4                      ; Restore R4 to its original self
  50    54   04   01   EF 0272         EXTZV   #1,#4,R4,R0             ; Extract a byte count
        50   10   50   83 0277         SUBB3   R0,#16,R0               ; Loop count is 16 minus byte count
                         027B
                         027B  ; Note that the loop count can never be zero because we are testing a 17-byte
                         027B  ; string and the largest output string can be 16 bytes long.
                         027B
                         027B         MARK_POINT      ADD_SUB_BSBW_24
                 71   95 027B  60$:   TSTB    -(R1)                   ; Check next byte for nonzero
                 04   12 027D         BNEQ    70$                     ; Nonzero means overflow has occurred
              F9 50   F5 027F         SOBGTR  R0,60$                  ; Check for end of this loop
                         0282
                      05 0282         RSB                             ; This is return path for no overflow
                         0283
            5B   02   88 0283  70$:   BISB    #PSL$M_V,R11            ; Indicate that overflow has occurred
                      05 0286         RSB                             ; ... and return to the caller
```

```
0287         .SUBTITLE       VAX$MULP - Multiply Packed
0287   ;+
0287   ; Functional Description:
0287   ;
0287   ;       The multiplicand string specified by the multiplicand length and
0287   ;       multiplicand address operands is multiplied by the multiplier string
0287   ;       specified by the multiplier length and multiplier address operands.  The
0287   ;       product string specified by the product length and product address
0287   ;       operands is replaced by the result.
0287   ;
0287   ; Input Parameters:
0287   ;
0287   ;       R0 - mulrlen.rw         Number of digits in multiplier string
0287   ;       R1 - mulraddr.ab        Address of multiplier string
0287   ;       R2 - muldlen.rw         Number of digits in multiplicand string
0287   ;       R3 - muldaddr.ab        Address of multiplicand string
0287   ;       R4 - prodlen.rw         Number of digits in product string
0287   ;       R5 - prodaddr.ab        Address of product string
0287   ;
0287   ; Output Parameters:
0287   ;
0287   ;       R0 = 0
0287   ;       R1 = Address of the byte containing the most significant digit of
0287   ;            the multiplier string
0287   ;       R2 = 0
0287   ;       R3 = Address of the byte containing the most significant digit of
0287   ;            the multiplicand string
0287   ;       R4 = 0
0287   ;       R5 = Address of the byte containing the most significant digit of
0287   ;            the string containing the product
0287   ;
0287   ; Condition Codes:
0287   ;
0287   ;       N <- product string LSS 0
0287   ;       Z <- product string EQL 0
0287   ;       V <- decimal overflow
0287   ;       C <- 0
0287   ;
0287   ; Register Usage:
0287   ;
0287   ;       This routine uses all of the general registers. The condition codes
0287   ;       are computed at the end of the instruction as the final result is
0287   ;       stored in the product string. R11 is used to record the condition
0287   ;       codes.
0287   ;
0287   ; Notes:
0287   ;
0287   ;    1. This routine uses a large amount of stack space to allow storage of
0287   ;       intermediate results in a convenient form. Specifically, each digit
0287   ;       pair of the longer input string is stored in binary in a longword on
0287   ;       the stack. In addition, 32 longwords are set aside to hold the product
0287   ;       intermediate result. Each longword contains a binary number between 0
0287   ;       and 99.
0287   ;
0287   ;       After the multiplication is complete. Each longword is removed from
0287   ;       the stack, converted to a packed decimal pair, and stored in the
0287   ;       output string. Any nonzero cells remaining on the stack after the
```

```
                              0287  ;       output string has been completely filled are the indication of decimal
                              0287  ;       overflow.
                              0287  ;
                              0287  ;       The purpose of this method of storage is to avoid decimal/binary or
                              0287  ;       even byte/longword conversions during the calculation of intermediate
                              0287  ;       results.
                              0287  ;
                              0287  ;    2. Trailing zeros are removed from the larger string. All zeros in
                              0287  ;       the shorter string are eliminated in the sense that no arithmetic
                              0287  ;       is performed. The output array pointer is simply advanced to point
                              0287  ;       to the next higher array element.
                              0287  ;-
                              0287
                              0287  VAX$MULP::
              0FFF 8F    BB   0287          PUSHR   #^M<R0,R1,R2,R3,R4,R5,R6,R7,R8,R9,R10,R11>      ; Save the lot
                              028B
                              028B          ESTABLISH_HANDLER       -       ; Store address of access
                              028B                  ARITH_ACCVIO            ;  violation handler
                              0290
                              0290          ROPRAND_CHECK   R4                      ; Insure that R4 is LEQU 31
                              029B
                              029B          ROPRAND_CHECK   R2                      ; Insure that R2 is LEQU 31
                              02A3          MARK_POINT      MULP_BSBW_0
                 FD5A'   30   02A3          BSBW    DECIMAL$STRIP_ZEROS_R2_R3       ; Strip high order zeros from R2/R3
                              02A6
                              02A6          ROPRAND_CHECK   R0                      ; Insure that R0 is LEQU 31
                              02AE          MARK_POINT      MULP_BSBW_0
                 FD4F'   30   02AE          BSBW    DECIMAL$STRIP_ZEROS_R0_R1       ; Strip high order zeros from R0/R1
                              02B1
   50    50   04    01   EF   02B1          EXTZV   #1,#4,R0,R0             ; Convert digit count to byte count
                    50   D6   02B6          INCL    R0                      ; Include least significant digit
                              02B8
   52    52   04    01   EF   02B8          EXTZV   #1,#4,R2,R2             ; Convert digit count to byte count
                    52   D6   02BD          INCL    R2                      ; Include least significant digit
                              02BF
              52    50   D1   02BF          CMPL    R0,R2                   ; See which string is larger
                    08   1A   02C2          BGTRU   3$                      ; R2/R3 describes the longer string
              58    52   7D   02C4          MOVQ    R2,R8                   ; R8 and R9 describe the longer string
              7E    50   7D   02C7          MOVQ    R0,-(SP)                ; Shorter string descriptor also saved
                    06   11   02CA          BRB     6$
                              02CC
              58    50   7D   02CC  3$:     MOVQ    R0,R8                   ; R8 and R9 describe the longer string
              7E    52   7D   02CF          MOVQ    R2,-(SP)                ; Shorter string descriptor also saved
                              02D2
                              02D2  ; Create space for the output array on the stack (32 longwords of zeros)
                              02D2
              50    08   D0   02D2  6$:     MOVL    #8,R0                   ; Eight pairs of quadwords
                              02D5
                    7E   7C   02D5  10$:    CLRQ    -(SP)                   ; Clear one pair
                    7E   7C   02D7          CLRQ    -(SP)                   ; ... and another
                 F9 50   F5   02D9          SOBGTR  R0,10$                  ; Do all eight pairs
                              02DC
              57    5E   D0   02DC          MOVL    SP,R7                   ; Store beginning of output array in R7
                              02DF
                              02DF  ; The longer input array will be stored on the stack as an array of
                              02DF  ; longwords. Each array element contains a number between 0 and 99,
                              02DF  ; representing a pair of digits in the original packed decimal string.
```

```
                              02DF  ; Because the units digit is stored with the sign in packed decimal format,
                              02DF  ; it is necessary to shift the number as we store it. This is accomplished by
                              02DF  ; multiplying the number by ten.
                              02DF  ;
                              02DF  ; The longer array is described by R8 (byte count) and R9 (address of most
                              02DF  ; significant digit pair).
                              02DF
         55    58    59   C1  02DF          ADDL3   R9,R8,R5                ; Point R5 beyond sign digit
               54    58   D0  02E3          MOVL    R8,R4                   ; R4 contains the loop count
                              02E6
                              02E6  ; An array of longwords is allocated on the stack. R3 starts out pointing
                              02E6  ; at the longword beyond the top of the stack. The first remainder, guaranteed
                              02E6  ; to be zero, is "stored" here. The rest of the digit pairs are stored safely
                              02E6  ; below the top of the stack.
                              02E6
               53    58   CE  02E6          MNEGL   R8,R3                   ; Stack grows toward lower addresses
            5E     6E43   DE  02E9          MOVAL   (SP)[R3],SP             ; Allocate the space
         53    5E    04   C3  02ED          SUBL3   #4,SP,R3                ; Point R3 at next lower longword
                              02F1
                              02F1          MARK_POINT      MULP_R8
               51    75   9A  02F1  20$:    MOVZBL  -(R5),R1                ; Get next digit pair
       51      0000'CF41  9A  02F4          MOVZBL  DECIMAL$PACKED_TO_BINARY_TABLE[R1],-
                              02FA                          R1              ; Convert digits to binary
    50    52    51    0A  7A  02FA          EMUL    #10,R1,R2,R0            ; Multiply by 10
 83    52    50  00000064 8F  7B  02FF      EDIV    #100,R0,R2,(R3)+        ; Divide by 100
                  E6 54   F5  0308          SOBGTR  R4,20$
                              030B
               63    52   D0  030B          MOVL    R2,(R3)                 ; Store final quotient
               59    5E   D0  030E          MOVL    SP,R9                   ; Remember array address in R9
                   6E48   DF  0311          PUSHAL  (SP)[R8]                ; Store start of fixed size area
                              0314
                              0314  ; Check for trailing zeros in the input array stored on the stack. If any are
                              0314  ; present, they are removed and the product array is adjusted accordingly.
                              0314
                     89   D5  0314  30$:    TSTL    (R9)+                   ; Is next number zero?
                     08   12  0316          BNEQ    40$                     ; Leave loop if nonzero
               57    04   C0  0318          ADDL    #4,R7                   ; Advance output pointer to next element
                  F6 58   F5  031B          SOBGTR  R8,30$                  ; Keep going
                              031E
                              031E  ; If we drop through the loop, then the entire input array is zero. There is
                              031E  ; no need to perform any arithmetic because the product will be zero (and the
                              031E  ; output array on the stack starts out as zero). The only remaining work is
                              031E  ; to store the result in the output string and set the condition codes.
                              031E
                     2C   11  031E          BRB     70$                     ; Exit to end processing
                              0320
                              0320  ; Now multiply the input array by each successive digit pair. In order to
                              0320  ; allow R10 to continue to locate ARITH_ACCVIO while we execute this loop, it
                              0320  ; is necessary to perform a small amount of register juggling. In essence,
                              0320  ; R8 and R9 switch the identity of the string that they describe.
                              0320
               59    04   C2  0320  40$:    SUBL    #4,R9                   ; Readjust input array pointer
               7E    58   7D  0323          MOVQ    R8,-(SP)                ; Save R8/R9 descriptor on stack
            58    08 AE   D0  0326          MOVL    8(SP),R8                ; Point R8 at start of 32-longword array
         58     0080 C8   7D  032A          MOVQ    <32*4>(R8),R8           ; Get descriptor that follows that array
               59    58   C0  032F          ADDL2   R8,R9                   ; Point R9 beyond sign byte
                              0332
```

```
       53    87   DE  0332  50$:    MOVAL   (R7)+,R3                   ; Output array address to R3
                      0335          MARK_POINT      MULP_AT_SP
       51    79   9A  0335          MOVZBL  -(R9),R1                   ; Next digit pair to R1
  56   0000'CF41  9A  0338          MOVZBL  DECIMAL$PACKED_TO_BINARY_TABLE[R1],-
                      033E                          R6                 ; Convert digits to binary
             06   13  033E          BEQL    60$                        ; Skip the work if zero
       54    6E   7D  0340          MOVQ    (SP),R4                    ; Input array descriptor to R4/R5
           0104   30  0343          BSBW    EXTEND_STRING_MULTIPLY     ; Do the work
          E9 58   F5  0346  60$:    SOBGTR  R8,50$                     ; Any more multiplier digits?
                      0349
       5E    08   C0  0349          ADDL    #8,SP                      ; Discard saved long string descriptor
                      034C
       5E    6E   D0  034C  70$:    MOVL    (SP),SP                    ; Remove input array from stack
                      034F
                      034F  ; At this point, the product string is located in a 32-longword array on
                      034F  ; the top of the stack. Each longword corresponds to a pair of digits in
                      034F  ; the output string. As digits are removed from the stack, they are checked
                      034F  ; for nonzero to obtain the correct setting of the Z-bit. After the output
                      034F  ; string has been filled, the remainder of the product string is removed from
                      034F  ; the stack. If a nonzero result is detected at this stage, the V-bit is set.
                      034F
       59    20   D0  034F          MOVL    #32,R9                     ; Set up array counter
  54   0098 CE    7D  0352          MOVQ    < <32*4> + -               ; Skip over 32-longword array
                      0357                <2*4>  + -                   ;  and saved string descriptor
                      0357                <4*4> >(SP),R4               ;  to retrieve original R4 and R5
```

```
                         0357        .SUBTITLE       Common Exit Path for VAX$MULP and VAX$DIVP
                         0357  ;+
                         0357  ; The code for VAX$MULP and VAX$DIVP merges at this point. The result is stored
                         0357  ; in an array of longwords at the top of the stack. The size of this array is
                         0357  ; stored in R9. The original R4 and R5 heve been retrieved from the stack.
                         0357  ;
                         0357  ; Input Parameters:
                         0357  ;
                         0357  ;       R4 - Contains byte count of destination string in R4 <1:4>
                         0357  ;       R5 - Address of most significant digit of destination string
                         0357  ;       R9 - Count of longwords in result array on stack
                         0357  ;
                         0357  ;       Contents of result array
                         0357  ;
                         0357  ; Implicit Input:
                         0357  ;
                         0357  ;       Signs of two input factors (multiplier and multiplicand or
                         0357  ;               divisor and dividend)
                         0357  ;-
                         0357
                         0357  MULTIPLY_DIVIDE_EXIT:
                 5B   DC 0357          MOVPSL  R11                     ; Get current PSL
  5B   04   00   04   F0 0359          INSV    #PSL$M_Z,#0,#4,R11      ; Clear all codes except Z-bit
                         035E          ESTABLISH_HANDLER       -       ; Store address of access
                         035E                  ARITH_ACCVIO            ;  violation handler again
  53   54   04   01   EF 0363          EXTZV   #1,#4,R4,R3             ; Excess byte count to R3
                 3B   13 0368          BEQL    125$                    ; Skip to single digit code
       57   55   53   C1 036A          ADDL3   R3,R5,R7                ; Remember address of sign byte
       55   57   01   C1 036E          ADDL3   #1,R7,R5                ; Point R5 beyond end of product string
                         0372
            51   8E   D0 0372  80$:    MOVL    (SP)+,R1                ; Remove next value from stack
                 03   13 0375          BEQL    90$                     ; Do not clear Z-bit if zero
            5B   04   8A 0377          BICB2   #PSL$M_Z,R11            ; Clear Z-bit
                         037A
                         037A          MARK_POINT      MULP_DIVP_R9
     75   0000'CF41   90 037A  90$:    MOVB    DECIMAL$BINARY_TO_PACKED_TABLE[R1],-
                         0380                  -(R5)                   ; Store converted sum byte
                 59   D7 0380          DECL    R9                      ; One less element on the stack
                 1C   15 0382          BLEQ    116$                    ; Exit loop if result array exhausted
             EB 53    F5 0384          SOBGTR  R3,80$                  ; Keep going?
                         0387
             22 54    E9 0387  100$:   BLBC    R4,120$                 ; Different for even digit count
                         038A
                         038A  ; The output string consists of an odd number of digits. A complete digit
                         038A  ; pair can be stored in the most significant (lowest addressed) byte of
                         038A  ; the product string.
                         038A
            51   8E   D0 038A          MOVL    (SP)+,R1                ; Remove next value from stack
                 03   13 038D          BEQL    110$                    ; Do not clear Z-bit if zero
            5B   04   8A 038F          BICB2   #PSL$M_Z,R11            ; Clear Z-bit
                         0392
                         0392          MARK_POINT      MULP_DIVP_R9
     75   0000'CF41   90 0392  110$:   MOVB    DECIMAL$BINARY_TO_PACKED_TABLE[R1],-
                         0398                  -(R5)                   ; Store converted sum byte
                 59   D7 0398          DECL    R9                      ; One less element on the stack
                 04   15 039A          BLEQ    116$                    ; Exit loop if result array exhausted
                 38   11 039C          BRB     140$                    ; Perform overflow check
```

```
                       039E
                       039E  ; This loop executes if the result array has fewer elements than the output
                       039E  ; string. The remaining bytes in the output string are filled with zeros.
                       039E  ; There is no need for an overflow check.
                       039E
                       039E          MARK_POINT      MULP_DIVP_8
              75   94  039E  114$:   CLRB    -(R5)                   ; Store another zero byte
           FB 53   F4  03A0  116$:   SOBGEQ  R3,114$                 ; Any more room in output string
                       03A3
              38   11  03A3          BRB     150$                    ; Determine sign of result
                       03A5
                       03A5  ; This code path is used in the case where the output digit count is 0 or 1.
                       03A5  ; R5 must be advanced
                       03A5
           57 55   D0  03A5  125$:   MOVL    R5,R7                   ; Remember address of output sign byte
              55   D6  03A8          INCL    R5                      ; Advance R5 so common code can be used
              DB   11  03AA          BRB     100$                    ; Join common code path
                       03AC
                       03AC  ; The output string consists of an even number of digits. Only the low order
                       03AC  ; nibble is stored in the most significant (lowest addresses) byte. A zero is
                       03AC  ; stored in the high order nibble. If the high order digit would have been
                       03AC  ; nonzero, the V-bit is set and the overflow check is bypassed because there
                       03AC  ; are faster ways to clean the stack if we do not have to check for nonzero
                       03AC  ; at the same time.
                       03AC
           51 8E   D0  03AC  120$:   MOVL    (SP)+,R1                ; Remove next value from stack
   51  0000'CF41   90  03AF          MOVB    DECIMAL$BINARY_TO_PACKED_TABLE[R1],-
                       03B5                          R1              ; Obtain converted sum byte
                       03B5          MARK_POINT      MULP_DIVP_R9
   75  51  F0 8F   8B  03B5          BICB3   #^XF0,R1,-(R5)          ; Store byte, clearing high order nibble
              03   13  03BA          BEQL    130$                    ; Do not clear Z-bit if zero
           5B 04   8A  03BC          BICB2   #PSL$M_Z,R11            ; Clear Z-bit
       51  F0 8F   93  03BF  130$:   BITB    #^XF0,R1                ; Is high order nibble nonzero?
              06   12  03C3          BNEQ    133$                    ; Yes, go set overflow bit
              59   D7  03C5          DECL    R9                      ; One less element on the stack
              D7   15  03C7          BLEQ    116$                    ; Exit loop if result array exhausted
              0B   11  03C9          BRB     140$                    ; Check rest of result array for nonzero
                       03CB
                       03CB  ; If we detect overflow, we need to adjust R9 to reflect the nonzero longword
                       03CB  ; removed from the stack before we enter the next code block that sets the
                       03CB  ; V-bit and cleans off the stack based on the contents of R9.
                       03CB
              59   D7  03CB  133$:   DECL    R9                      ; One more longword removed from stack
                       03CD
                       03CD  ; A nonzero digit has been discovered in a position that cannot be stored in
                       03CD  ; the output string. Set the V-bit, remove the rest of the product array from
                       03CD  ; the stack, and join the exit processing in the code that determines the sign
                       03CD  ; of the product.
                       03CD
           5B 02   88  03CD  135$:   BISB    #PSL$M_V,R11            ; Set the overflow bit
         5E 6E49   DE  03D0          MOVAL   (SP)[R9],SP             ; Clean off remaining product string
              07   11  03D4          BRB     150$                    ; Go to code that determines the sign
                       03D6
                       03D6  ; The remainder of the product array must be removed from the stack. A nonzero
                       03D6  ; result causes the V-bit to be set and the rest of the loop to be skipped.
                       03D6  ; Note that there is always a nonzero loop count remaining at this point.
                       03D6
```

```
                     8E   D5  03D6  140$:    TSTL    (SP)+                         ; Is next longword zero?
                     F1   12  03D8           BNEQ    133$                          ; No, leave loop
                  F9 59   F5  03DA           SOBGTR  R9,140$
                              03DD
                              03DD  ; The final product string has been stored and the V- and Z-bits have their
                              03DD  ; correct settings. The sign of the product must be determined from the
                              03DD  ; signs of the two input strings. Opposite signs produce a negative product.
                              03DD  ; Same signs (in any representation) produce a plus sign in the output string.
                              03DD
              5E     08   C0  03DD  150$:    ADDL    #8,SP                         ; Discard saved string descriptor
              56     0C   D0  03E0           MOVL    #12,R6                        ; Assume final result is positive
              50     6E   7D  03E3           MOVQ    (SP),R0                       ; Retrieve original R0/R1 pair
  50    50    04     01   EF  03E6           EXTZV   #1,#4,R0,R0                   ; Get byte count for first input string
              51     50   C0  03EB           ADDL    R0,R1                         ; Point R1 to byte containing sign
                              03EE           MARK_POINT      MULP_DIVP_0
     50    61     F0 8F   8B  03EE           BICB3   #^B11110000,(R1),R0           ; R0 contains the sign 'digit"
                              03F3
                              03F3           CASE    R0,TYPE=B,LIMIT=#10,<-        ; Dispatch on sign digit
                              03F3                   220$,-                        ; 10 => sign is "+"
                              03F3                   210$,-                        ; 11 => sign is "-"
                              03F3                   220$,-                        ; 12 => sign is "+"
                              03F3                   210$,-                        ; 13 => sign is "-"
                              03F3                   220$,-                        ; 14 => sign is "+"
                              03F3                   220$,-                        ; 15 => sign is "+"
                              03F3                   >
                              0403
              54     01   D0  0403  210$:    MOVL    #1,R4                         ; Count a minus sign
                     02   11  0406           BRB     230$                          ; Now check second input sign
                              0408
                     54   D4  0408  220$:    CLRL    R4                            ; No real minus signs so far
                              040A
           52     08 AE   7D  040A  230$:    MOVQ    8(SP),R2                      ; Retrieve original R2/R3 pair
  52    52    04     01   EF  040E           EXTZV   #1,#4,R2,R2                   ; Get byte count for second input string
              53     52   C0  0413           ADDL    R2,R3                         ; Point R3 to byte containing sign
                              0416           MARK_POINT      MULP_DIVP_0
     52    63     F0 8F   8B  0416           BICB3   #^B11110000,(R3),R2           ; R2 contains the sign "digit"
                              041B
                              041B           CASE    R2,TYPE=B,LIMIT=#10,<-        ; Dispatch on sign digit
                              041B                   250$,-                        ; 10 => sign is "+"
                              041B                   240$,-                        ; 11 => sign is "-"
                              041B                   250$,-                        ; 12 => sign is "+"
                              041B                   240$,-                        ; 13 => sign is "-"
                              041B                   250$,-                        ; 14 => sign is "+"
                              041B                   250$,-                        ; 15 => sign is "+"
                              041B                   >
                              042B
                     54   D6  042B  240$:    INCL    R4                            ; Remember that sign was minus
                  09 54   E9  042D  250$:    BLBC    R4,260$                       ; Even parity indicates positive result
           10 5B     02   E0  0430           BBS     #PSL$V_Z,R11,270$             ; Step out of line for minus zero check
              5B     08   88  0434           BISB    #PSL$M_N,R11                  ; Set N-bit in saved PSW
                     56   D6  0437  255$:    INCL    R6                            ; Change sign to minus
                              0439
                              0439           MARK_POINT      MULP_DIVP_0
  67    04    00     56   F0  0439  260$:    INSV    R6,#0,#4,(R7)                 ; Store sign in result string
                  10 AE   D4  043E           CLRL    16(SP)                        ; Set saved R4 to zero
                  FBBC'   31  0441           BRW     VAX$DECIMAL_EXIT              ; Join common exit code
                              0444
```

```
                  0444  1493 ; If the result is negative zero, then it must be changed to positive zero
                  0444  1494 ; unless overflow has occurred, in which case, the sign is left as negative
                  0444  1495 ; but the N-bit is clear.
                  0444  1496
EF 5B   01   E0   0444  1497 270$:    BBS     #PSL$V_V,R11,255$        ; Make sign negative if overflow
        EF   11   0448  1498          BRB     260$                     ; Sign will be positive
```

```
044A          .SUBTITLE       EXTEND_STRING_MULTIPLY - Multiply a String by a Number
044A     ;+
044A     ; Functional Description:
044A     ;
044A     ;       This routine multiplies an array of numbers (each array element LEQU
044A     ;       99) by a number (also LEQU 99). The resulting product array is added
044A     ;       to another array, each of whose elements is also LEQU 99.
044A     ;
044A     ; Input Parameters:
044A     ;
044A     ;       R3 - Pointer to output array
044A     ;       R4 - Input array size
044A     ;       R5 - Input array address
044A     ;       R6 - Multiplier
044A     ;
044A     ; Output Parameters:
044A     ;
044A     ;       None
044A     ;
044A     ; Implicit Output:
044A     ;
044A     ;       The output array is altered.
044A     ;
044A     ;       An intermediate product array is produced by multiplying each input
044A     ;       array element by the multiplier. Each product array element is then
044A     ;       added to the corresponding output array element.
044A     ;
044A     ; Side Effects:
044A     ;
044A     ;       R3, R4, and R5 are modified by this routine.
044A     ;
044A     ;       R6 is preserved.
044A     ;
044A     ;       R0, R1, and R2 are used as scratch registers. R0 and R1 contain the
044A     ;       quadword result of EMUL that is then passed into EDIV.
044A     ;
044A     ; Assumptions:
044A     ;
044A     ;       This routine assumes that all array elements lie in the range from 0
044A     ;       to 99 inclusive. (This is true if all input strings contain only legal
044A     ;       decimal digits.) The arithmetic performed by this routine will
044A     ;       maintain this assumption. That is,
044A     ;
044A     ;                           input array element    LEQU 99
044A     ;               times                multiplier    LEQU 99
044A     ;                      -------------------------
044A     ;                                       product                 LEQU 99*99
044A     ;               plus                      carry    LEQU 99
044A     ;                      -------------------------
044A     ;                              modified product                 LEQU 99*100
044A     ;               plus   old output array element    LEQU 99
044A     ;                      -------------------------
044A     ;                      new output array element                 LEQU 99*101 = 9999
044A     ;
044A     ;       A number LEQU 9999, when divided by 100, is guaranteed to produce both
044A     ;       a quotient and a remainder LEQU 99.
044A     ;-
```

```
                                044A
                                044A  EXTEND_STRING_MULTIPLY:
                    52     D4   044A          CLRL    R2                          ; Initial carry is zero
                                044C
         50   52   85   56     7A   044C  10$:    EMUL    R6,(R5)+,R2,R0              ; Form modified product (R0 LEQU 9900)
                   50   63     C0   0451          ADDL2   (R3),R0                     ; Add old output array element
 83   52   50  00000064 8F     7B   0454          EDIV    #100,R0,R2,(R3)+            ; Remainder to output array
                                045D                                              ; Quotient becomes carry
                     EC 54     F5   045D          SOBGTR  R4,10$                      ; Keep going?
                                0460
                                0460  ; This remaining code looks more complicated than it actually is. In the
                                0460  ; usual case, the routine exits immediately. In the event that a carry
                                0460  ; occurs, one additional entry in the output array will be modified. Only in
                                0460  ; the rare case of an output array consisting of a string of 99s will any
                                0460  ; significant looping occur.
                                0460
                   63   52     C0   0460          ADDL2   R2,(R3)                     ; Add final carry
         00000064 8F    63     D1   0463  20$:    CMPL    (R3),#100                   ; Do we overflow into next digit pair?
                        01     1E   046A          BGEQU   30$                         ; Branch if carry
                        05     046C               RSB                                 ; Otherwise, all done
                                046D
         83   00000064 8F      C2   046D  30$:    SUBL    #100,(R3)+                  ; Readjust entry and advance pointer
                        63     D6   0474          INCL    (R3)                        ; Propogate carry
                        EB     11   0476          BRB     20$                         ; ... and test this entry for overflow
```

```
0478            .SUBTITLE       VAX$DIVP - Divide Packed
0478    ;+
0478    ; Functional Description:
0478    ;
0478    ;       The dividend string  specified  by  the  dividend  length  and  dividend
0478    ;       address  operands  is  divided  by  the  divisor string specified by the
0478    ;       divisor length  and  divisor  address  operands.   The  quotient  string
0478    ;       specified  by  the  quotient  length  and  quotient  address operands is
0478    ;       replaced by the result.
0478    ;
0478    ; Input Parameters:
0478    ;
0478    ;       R0 - divrlen.rw         Number of digits in divisor string
0478    ;       R1 - divraddr.ab        Address of divisor string
0478    ;       R2 - divdlen.rw         Number of digits in dividend string
0478    ;       R3 - divdaddr.ab        Address of dividend string
0478    ;       R4 - quolen.rw          Number of digits in quotient string
0478    ;       R5 - quoaddr.ab         Address of quotient string
0478    ;
0478    ; Output Parameters:
0478    ;
0478    ;       R0 = 0
0478    ;       R1 = Address of the byte containing the most significant digit of
0478    ;            the divisor string
0478    ;       R2 = 0
0478    ;       R3 = Address of the byte containing the most significant digit of
0478    ;            the dividend string
0478    ;       R4 = 0
0478    ;       R5 = Address of the byte containing the most significant digit of
0478    ;            the string containing the quotient
0478    ;
0478    ; Condition Codes:
0478    ;
0478    ;       N <- quotient string LSS 0
0478    ;       Z <- quotient string EQL 0
0478    ;       V <- decimal overflow
0478    ;       C <- 0
0478    ;
0478    ; Register Usage:
0478    ;
0478    ;       This routine uses all of the general registers. The condition codes
0478    ;       are computed at the end of the instruction as the final result is
0478    ;       stored in the quotient string. R11 is used to record the condition
0478    ;       codes.
0478    ;
0478    ; Algorithm:
0478    ;
0478    ;       This algorithm is the straightforward approach described in
0478    ;
0478    ;               The Art of Computer Programming
0478    ;               Second Edition
0478    ;
0478    ;               Volume 2 / Seminumerical Algorithms
0478    ;               Donald E. Knuth
0478    ;
0478    ;               1981
0478    ;               Addison-Wesley Publishing Company
```

```
                       0478  ;              Reading, Massachusetts
                       0478  ;
                       0478  ; Notes:
                       0478  ;
                       0478  ;      The choice of a longword array to store the auotient deserves a
                       0478  ;      comment. In VAX$MULP, a longword array was used because its elements
                       0478  ;      were used directly by MULP and DIVP instructions. The use of longwords
                       0478  ;      eliminated the need to convert back and forth between longwords and
                       0478  ;      bytes. In this routine, the QUOTIENT_DIGIT routine returns its result
                       0478  ;      in a register, which result can easily be stored in whatever way is
                       0478  ;      convenient. By using longwords instead of bytes, this routine can use
                       0478  ;      the same end processing code as MULP, a sizeable savings in code.
                       0478  ;-
                       0478
                       0478         .ENABLE         LOCAL_BLOCK
                       0478
                       0478  ;+
                       0478  ; This code path is entered if the divisor is zero.
                       0478  ;
                       0478  ; Input Parameter:
                       0478  ;
                       0478  ;      (SP) - Return PC
                       0478  ;
                       0478  ; Output Parameters:
                       0478  ;
                       0478  ;      0(SP) - SRM$K_FLT_DIV_T (Arithmetic trap code)
                       0478  ;      4(SP) - Final state PSL
                       0478  ;      8(SP) - Return PC
                       0478  ;
                       0478  ; Implicit Output:
                       0478  ;
                       0478  ;      Control passes through this code to VAX$REFLECT_TRAP.
                       0478  ;-
                       0478
                       0478  DIVIDE_BY_ZERO:
   0FFF 8F      BA     0478         POPR    #^M<R0,R1,R2,R3,R4,R5,R6,R7,R8,R9,R10,R11>
                       047C                                         ; Restore registers and reset SP
        7E      DC     047C         MOVPSL  -(SP)                   ; Save final PSL on stack
        04      DD     047E         PUSHL   #SRM$K_FLT_DIV_T        ; Store arithmetic trap code
     FB7D'      31     0480         BRW     VAX$REFLECT_TRAP        ; Report exception
                       0483
                       0483  ; If the divisor contains more nonzero digits than the dividend, then the
                       0483  ; quotient will be identically zero. Set up the stack and the registers (R4,
                       0483  ; R5, and R9)  so that the exit code will be entered to produce this result.
                       0483
        7E      D4     0483  1$:    CLRL    -(SP)                   ; Fake a quotient digit
     59 01      D0     0485         MOVL    #1,R9                   ; Count that digit
      FECC      31     0488         BRW     MULTIPLY_DIVIDE_EXIT    ; Store the zero in the output string
                       048B
                       048B  VAX$DIVP::
   0FFF 8F      BB     048B         PUSHR   #^M<R0,R1,R2,R3,R4,R5,R6,R7,R8,R9,R10,R11>      ; Save the lot
                       048F
                       048F         ESTABLISH_HANDLER      -        ; Store address of access
                       048F                 ARITH_ACCVIO            ;  violation handler
                       0494
                       0494         ROPRAND_CHECK   R4                      ; Insure that R4 is LEQU 31
                       049F
```

```
                          049F         ROPRAND_CHECK   R2                   ; Insure that R2 is LEQU 31
                          04A7         MARK_POINT      DIVP_BSBW_0
              FB56'   30  04A7         BSBW    DECIMAL$STRIP_ZEROS_R2_R3    ; Strip high order zeros from R2/R3
                          04AA
                          04AA         ROPRAND_CHECK   R0                   ; Insure that R0 is LEQU 31
                          04B2         MARK_POINT      DIVP_BSBW_0
              FB4B'   30  04B2         BSBW    DECIMAL$STRIP_ZEROS_R0_R1    ; Strip high order zeros from R0/R1
                          04B5
                          04B5  ; Insure that the divisor is not zero. Because leading zeros have already
                          04B5  ; been eliminated, the divisor can only be zero if R0 is 0 (zero length
                          04B5  ; strings are identically zero) or 1 (R1 contains a sign digit in the low
                          04B5  ; order nibble and zero in the high order nibble). Note that an exception
                          04B5  ; will not be generated if an even length string has an illegal nonzero digit
                          04B5  ; stored in its most significant nibble (including an illegal form of a zero
                          04B5  ; length string.
                          04B5
   50   50   04   01   EF 04B5         EXTZV   #1,#4,R0,R0          ; Convert divisor digit count to bytes
                  06   12 04BA         BNEQ    10$                  ; Skip zero divisor check unless zero
                          04BC         MARK_POINT      DIVP_0
        61   F0 8F    93  04BC         BITB    #^B11110000,(R1)     ; Check for zero in ones digit
                  B6   13 04C0         BEQL    DIVIDE_BY_ZERO       ; Generate exception if zero
                          04C2
                          04C2  ; This routine chooses to do its work with a fair amount of internal storage,
                          04C2  ; all of it allocated on the stack. The quotient is stored as it is computed,
                          04C2  ; in a 16-longword array. The dividend and divisor are stored as longword arrays,
                          04C2  ; with each array element storing a digit pair from the original packed
                          04C2  ; decimal string. The numerator digits are shifted by one digit (multiplied
                          04C2  ; by ten) so that the quotient has its digits correctly placed, leaving room
                          04C2  ; for a sign in the low order nibble of the least significant byte. A scratch
                          04C2  ; array is also allocated on the stack to accommodate intermediate results
                          04C2  ; of the QUOTIENT_DIGIT routine.
                          04C2
                  50   D6 04C2  10$:   INCL    R0                   ; Include least significant digit
             58   50   7D 04C4         MOVQ    R0,R8                ; Let R8 and R9 describe the divisor
                          04C7
   52   52   04   01   EF 04C7         EXTZV   #1,#4,R2,R2          ; Convert dividend digit count to bytes
                  52   D6 04CC         INCL    R2                   ; Include least significant digit
             7E   52   7D 04CE         MOVQ    R2,-(SP)             ; Save dividend descriptor on stack
                          04D1
        56   52   50   C3 04D1         SUBL3   R0,R2,R6             ; Calculate main loop count
                  AC   1F 04D5         BLSSU   1$                   ; Quotient will be zero
                  56   D6 04D7         INCL    R6                   ; One extra digit is always there
                          04D9
                          04D9  ; Allocate R6 longwords of zero on the stack
                          04D9
             50   56   D0 04D9         MOVL    R6,R0                ; Let R0 be the loop counter
                  7E   D4 04DC  15$:   CLRL    -(SP)                ; Set aside another quotient digit
              FB  50   F5 04DE         SOBGTR  R0,15$               ; Keep going
                          04E1
             57   5E   D0 04E1         MOVL    SP,R7                ; Remember where this array starts
                          04E4
                          04E4  ; The divisor will be stored on the stack as an array of
                          04E4  ; longwords. Each array element contains a number between 0 and 99,
                          04E4  ; representing a pair of digits in the original packed decimal string.
                          04E4  ; Because the units digit is stored with the sign in packed decimal format,
                          04E4  ; it is necessary to shift the number as we store it. This is accomplished by
                          04E4  ; multiplying the number by ten.
```

```
                            04E4  ;
                            04E4  ; The divisor string is described by R8 (byte count) and R9 (address of most
                            04E4  ; significant digit pair).
                            04E4
          55   58   59  C1  04E4          ADDL3   R9,R8,R5                ; Point R5 beyond sign digit
               54   58  D0  04E8          MOVL    R8,R4                   ; R4 contains the loop count
                            04EB
                            04EB  ; Put in an extra digit place for the divisor. This allows several common
                            04EB  ; subroutines to be used when operating on the divisor string.
                            04EB
                    7E  D4  04EB          CLRL    -(SP)                   ; Set aside a place holder
                            04ED
                            04ED  ; An array of longwords is allocated on the stack. R3 starts out pointing
                            04ED  ; at the longword beyond the top of the stack. The first remainder, guaranteed
                            04ED  ; to be zero, is "stored" here. The rest of the digit pairs are stored safely
                            04ED  ; below the top of the stack.
                            04ED
               53   58  CE  04ED          MNEGL   R8,R3                   ; Stack grows toward lower addresses
            5E    6E43  DE  04F0          MOVAL   (SP)[R3],SP             ; Allocate the space
          53   5E   04  C3  04F4          SUBL3   #4,SP,R3                ; Point R3 at next lower longword
                            04F8
                            04F8          MARK_POINT      DIVP_R6_R7
               51   75  9A  04F8  20$:    MOVZBL  -(R5),R1                ; Get next digit pair
        51   0000'CF41  9A  04FB          MOVZBL  DECIMAL$PACKED_TO_BINARY_TABLE[R1],-
                            0501                  R1                      ; Convert digits to binary
     50   52   51   0A  7A  0501          EMUL    #10,R1,R2,R0            ; Multiply by 10
83   52   50  00000064  8F  7B  0506      EDIV    #100,R0,R2,(R3)+        ; Divide by 100
               E6   54  F5  050F          SOBGTR  R4,20$
                            0512
                            0512  ; There are two cases where the final quotient (contents of R2) is zero.
                            0512  ; In these cases, the number of nonzero digit pairs in the divisor array is
                            0512  ; smaller by one than the number of bytes containing the original packed decimal
                            0512  ; string. One case is a divisor string with an even number of digits. The
                            0512  ; second case is a divisor string with an odd number of digits but the most
                            0512  ; significant digit is zero (essentially a variant of the first case). The
                            0512  ; simplest way to handle all of these cases is to decrement R8, the divisor
                            0512  ; counter, if R2 is zero. Note that previous checks for a zero divisor
                            0512  ; prevent R8 from going to zero.
                            0512
               63   52  D0  0512          MOVL    R2,(R3)                 ; Store final quotient
                    0A  12  0515          BNEQ    25$                     ; Leave well enough alone if nonzero
                    56  D6  0517          INCL    R6                      ; One more quotient digit
               57   04  C2  0519          SUBL    #4,R7                   ; Make room for it
                    58  D7  051C          DECL    R8                      ; Count one less divisor "digit"
                    01  12  051E          BNEQ    25$
                            0520
                            0520  ;;; ********** BEGIN TEMP **********
                            0520  ;;;
                            0520  ;;; THE FOLLOWING HALT INSTRUCTION SHOULD BE REPLACED WITH THE CORRECT
                            0520  ;;; ABORT CODE.
                            0520  ;;;
                            0520  ;;; THE HALT IS SIMILAR TO THE
                            0520  ;;;
                            0520  ;;;     MICROCODE CANNOT GET HERE
                            0520  ;;;
                            0520  ;;; ERRORS THAT OTHER IMPLEMENTATIONS USE.
                            0520  ;;;
```

```
                    00  0520          halt                            ; This will cause an OPCDEC exception
                        0521  ;;;
                        0521  ;;; *********** END TEMP ***********
                        0521
         59    5E   D0  0521  25$:    MOVL    SP,R9                   ; R9 locates low order divisor digit
                        0524
                        0524  ; The dividend is stored on the stack as an array of longwords. It does not
                        0524  ; have its digit pairs shifted so that this storage loop is simpler. An extra
                        0524  ; place is set aside in the event that it is necessary to normalize the
                        0524  ; dividend and divisor before division is attempted.
                        0524
               7E   D4  0524          CLRL    -(SP)                   ; Set aside space for U[0]
      52     6746   DE  0526          MOVAL   (R7)[R6],R2             ; Retrieve dividend descriptor
         52    62   7D  052A          MOVQ    (R2),R2                 ; ... in two steps
                        052D
                        052D          MARK_POINT      DIVP_R6_R7
         51    83   9A  052D  30$:    MOVZBL  (R3)+,R1                ; Get next decimal digit pair
   7E   0000'CF41   9A  0530          MOVZBL  DECIMAL$PACKED_TO_BINARY_TABLE[R1],-
                        0536                  -(SP)                   ; Convert digits to binary
            F4 52   F5  0536          SOBGTR  R2,30$                  ; Loop through entire input string
                        0539
                        0539  ; From this point until the common exit path for MULP and DIVP is entered,
                        0539  ; no access violations that need to be backed out can occur. We do not need
                        0539  ; to keep the address of ARITH_ACCVIO in R10 for this stretch of code. Note
                        0539  ; that R10 must be reloaded before the exit code executes because the
                        0539  ; destination string is written and may cause access violations.
                        0539
      5A     6746   D0  0539          MOVL    (R7)[R6],R10            ; Retrieve size of dividend array
         5B    5E   D0  053D          MOVL    SP,R11                  ; R11 locates low order dividend digit
                        0540
                        0540  ; Allocate a scratch array on the stack the same size as the divisor array
                        0540  ; (which is one larger than the number of digit pairs)
                        0540
         52    58   CE  0540          MNEGL   R8,R2                   ; Need a negative index
     5E   FC AE42   DE  0543          MOVAL   -4(SP)[R2],SP           ; Adjust stack pointer
                        0548
                        0548  ;+
                        0548  ; At this point, the stack and relevant general registers contain the
                        0548  ; following information. In this description, N represents the number
                        0548  ; of digit pairs in the divisor and M represents the number of digit
                        0548  ; pairs in the dividend.
                        0548  ;
                        0548  ;                       +-----------------------------+ <-- SP
                        0548  ;     scratch           :        N+1 longwords        :
                        0548  ;                       +-----------------------------+ <-- R11
                        0548  ;     dividend          :        M+1 longwords        :
                        0548  ;                       +-----------------------------+ <-- R9
                        0548  ;     divisor           :        N+1 longwords        :
                        0548  ;                       +-----------------------------+ <-- R7
                        0548  ;     quotient          :      M+1-N longwords        :
                        0548  ;                       +-----------------------------+
                        0548  ;                       :          R0..R11            :
                        0548  ;                       +-----------------------------+
                        0548  ;
                        0548  ;       R6  - Number of longwords in quotient array (M+1-N)
                        0548  ;       R7  - Address of beginning of quotient array
                        0548  ;       R8  - Number of digit pairs in divisor (called N)
```

```
                        0548  ;       R9  - Address of low order digits in divisor
                        0548  ;       R10 - Number of digit pairs in dividend (called M)
                        0548  ;       R11 - Address of low order digits in dividend
                        0548  ;-
                        0548
               6E   DF  0548          PUSHAL  (SP)                    ; Store address of scratch array
          7E   58   7D  054A          MOVQ    R8,-(SP)                ; Remember divisor descriptor
          7E   5A   7D  054D          MOVQ    R10,-(SP)               ; Remember dividend descriptor
                        0550
                        0550  ; The algorithm that guesses the quotient digit can be guaranteed to be off
                        0550  ; by no more than two if the high order digit of the divisor (called V[1]) is
                        0550  ; at least as large as 50 (our radix divided by 2). If the high order digit
                        0550  ; is too small, we "normalize" the numerator and denomonator by multiplying
                        0550  ; them by the same number, namely 100/(V[1]+1).
                        0550
  50   FC A948     01   C1  0550          ADDL3   #1,-4(R9)[R8],R0        ; Compute V[1] + 1
               33  50   D1  0556          CMPL    R0,#51                  ; Compare to 50 + 1
                   14   18  0559          BGEQ    40$                     ; Skip normalization if V[1] big enough
53   00000064 8F   50   C7  055B          DIVL3   R0,#100,R3              ; Compute normalization factor
               54  58   7D  0563          MOVQ    R8,R4                   ; Get descriptor of divisor
                 00E0   30  0566          BSBW    MULTIPLY_STRING         ; Normalize divisor
               54  5A   7D  0569          MOVQ    R10,R4                  ; Get descriptor of dividend
                 00DA   30  056C          BSBW    MULTIPLY_STRING         ; Normalize dividend
                        056F
                        056F  ; We have now reached the point where we can start calculating quotient digits.
                        056F  ; In the following loop, R5 and R6 are loop invariants. R5 contains the number
                        056F  ; of digit pairs in the divisor. R6 always points to the longword beyond the
                        056F  ; most significant digit in the dividend string. R7 and R8 must be loaded on
                        056F  ; each pass through because these two pointers are modified. Notice that the
                        056F  ; address of the divisor array is exactly what we want to store in R6.
                        056F
               5A  56   7D  056F  40$:    MOVQ    R6,R10                  ; Let R10/R11 describe quotient and loop
                   5B   DD  0572          PUSHL   R11                     ; Save quotient address for exit code
           5B    6B4A   DE  0574          MOVAL   (R11)[R10],R11          ; Store quotient digits from high end
                        0578
                        0578  ; This rather harmless looking loop is where the work is done
                        0578
               55  58   7D  0578          MOVQ    R8,R5                   ; Initialize count and dividend address
               59  5A   D0  057B          MOVL    R10,R9                  ; Remember the loop count in R9
                        057E
            57   10 AE  7D  057E  50$:    MOVQ    16(SP),R7               ; Load divisor and scratch addresses
                 001F   30  0582          BSBW    QUOTIENT_DIGIT          ; Get the next quotient digit
               7B  53   D0  0585          MOVL    R3,-(R11)               ; Store it
               56  04   C2  0588          SUBL    #4,R6                   ; "Advance" dividend pointer
                F0 5A   F5  058B          SOBGTR  R10,50$                 ; ... and go gack for more
                        058E
                        058E  ; The quotient digits have been stored on the stack. Eliminate the rest of the
                        058E  ; stack storage and enter the completion code that this routine shares with
                        058E  ; VAX$MULP. Note that R9 is already set up with the longword count used by
                        058E  ; the exit code. Note also that R11 is pointing to the saved dividend descriptor
                        058E  ; that sits on top of the saved register array.
                        058E
               5E  6E   D0  058E          MOVL    (SP),SP                 ; Reset stack pointer
        54    18 AB49   DE  0591          MOVAL   < <4*2> + -             ; Skip over saved dividend descriptor
                        0596                      <4*4> >(R11)[R9],R4     ;  and retrieve original R4 and R5
               54  64   7D  0596          MOVQ    (R4),R4                 ; ... in two steps
                        0599
```

```
                        0599  ; The following is a HACK.
                        0599  ;
                        0599  ; The method used to obtain quotient digits generally leaves garbage (nonzero)
                        0599  ; in what will become the sign digit. (In fact, this is the tenths digit of a
                        0599  ; decimal expansion of the remainder.) We need to make the least significant
                        0599  ; digit a multiple of ten.
                        0599
   50   6E   0A   C7    0599          DIVL3   #10,(SP),R0             ; Divide by ten, losing remainder
   6E   50   0A   C5    059D          MULL3   #10,R0,(SP)             ; Store only tens digit
                        05A1
           FDB3   31    05A1          BRW     MULTIPLY_DIVIDE_EXIT    ; Join common exit code
                        05A4
                        05A4          .DISABLE        LOCAL_BLOCK
```

```
05A4          .SUBTITLE       QUOTIENT_DIGIT - Get Next Digit in Quotient
05A4    ;+
05A4    ; Functional Description:
05A4    ;
05A4    ;       This routine divides an (N+1)-element array of longwords by an N-element
05A4    ;       array, producing a single quotient digit in the range of 0 to 99
05A4    ;       inclusive. The dividend array is modified by subtracting the product
05A4    ;       of the divisor array and the quotient digit.
05A4    ;
05A4    ;       The "numbers" that this array operates on multiple precision numbers
05A4    ;       in radix 100. Each digit (a number between 0 and 99) is stored in a
05A4    ;       longword array element with more significant digits stored at higher
05A4    ;       addresses. The dividend string and the scratch string (also called the
05A4    ;       product string) contain one more element than the divisor string.
05A4    ;
05A4    ; Input Parameters:
05A4    ;
05A4    ;       R5 - Number of "digits" (array elements) in divisor array (preserved)
05A4    ;       R6 - Address of longword immediately following most significant
05A4    ;               digit of dividend string (preserved)
05A4    ;       R7 - Address of least significant digit in divisor string (modified)
05A4    ;       R8 - Address of least significant digit in product string (modified)
05A4    ;
05A4    ; Output Parameters:
05A4    ;
05A4    ;       R3 - The quotient that results from dividing the dividend string
05A4    ;               by the divisor string.
05A4    ;
05A4    ;       The final states of the three pointer registers are listed here
05A4    ;       for completeness.
05A4    ;
05A4    ;       R6 - Address of longword immediately following most significant
05A4    ;               digit of dividend string
05A4    ;
05A4    ;       R7 - Address of longword immediately following most significant digit
05A4    ;               of divisor string. This longword must always contain zero.
05A4    ;
05A4    ;       R8 - Address of longword immediately following most significant
05A4    ;               digit of product string
05A4    ;
05A4    ; Implicit Output:
05A4    ;
05A4    ;       The contents of the dividend array are modified to reflect the
05A4    ;       subtraction of the product string. The result of this subtraction
05A4    ;       could be stored elsewhere. It is a convenience to store it in the
05A4    ;       dividend array on top of those array elements that are no longer
05A4    ;       needed.
05A4    ;
05A4    ;       The contents of the divisor array are preserved.
05A4    ;
05A4    ; Side Effects:
05A4    ;
05A4    ;       R7 and R8 are modified by this routine. (See implicit output list.)
05A4    ;
05A4    ;       R5 and R6 are preserved.
05A4    ;
05A4    ;       R0, R1, R2, and R4 are used as scratch registers. R0 and R1 contain the
```

```
                                   05A4 ;       quadword result of EMUL that is then passed into EDIV. R2 is the
                                   05A4 ;       carry from one step to the next. R4 is the loop counter.
                                   05A4 ;-
                                   05A4
                                   05A4 QUOTIENT_DIGIT:
 F8 A6   FC A6   00C00064 8F   7A  05A4         EMUL    #100,-4(R6),-8(R6),R0   ; R0 <- 100 * U[j] + U[j+1]
                          50       05AE
            50   FC A745       C6  05AF         DIVL2   -4(R7)[R5],R0           ; R0 <- R0 / V[1]
                   53   50     D0  05B4         MOVL    R0,R3                   ; Store quotient "digit" in R3
                        65     13  05B7         BEQL    45$                     ; Nothing to do if quotient is zero
         00000064 8F    53     D1  05B9         CMPL    R3,#100                 ; Is quotient LEQU 99?
                        07     1F  05C0         BLSSU   5$                      ; Branch if quotient OK
         53   00000063 8F      D0  05C2         MOVL    #99,R3                  ; Otherwise start with 99
                                   05C9
                                   05C9 ; We will now multiply the divisor array by the quotient digit, storing the
                                   05C9 ; product in the scratch array.
                                   05C9
                        52     D4  05C9 5$:     CLRL    R2                      ; Start out with a carry of zero
                   54   55     D0  05CB         MOVL    R5,R4                   ; R4 will be the loop counter
                                   05CE
         50   52   87   53     7A  05CE 10$:    EMUL    R3,(R7)+,R2,R0          ; Multiply next divisor digit
 88   52   50   00000064 8F    7B  05D3         EDIV    #100,R0,R2,(R8)+        ; Remainder to input array
                                   05DC                                         ; Quotient becomes carry
                     EF 54     F5  05DC         SOBGTR  R4,10$                  ; More divisor digits?
                                   05DF
                   88   52     D0  05DF         MOVL    R2,(R8)+                ; Store final carry
                                   05E2
                                   05E2 ; If the product array is larger than the dividend array, then the quotient is
                                   05E2 ; too large. To avoid a second trip through the rather costly EMUL/EDIV loop,
                                   05E2 ; and also to avoid array subtraction that produces a negative result, we will
                                   05E2 ; first compare the product and dividend arrays. If the product is smaller, we
                                   05E2 ; can safely subtract. If the product is larger, we decrease the quotient by
                                   05E2 ; one and subtract the divisor array from the product array.
                                   05E2
                   50   56     D0  05E2 15$:    MOVL    R6,R0                   ; Point R0 and R1 to high address ends
                   51   58     D0  05E5         MOVL    R8,R1                   ; ... of dividend and scratch strings
                   54   55     D0  05E8         MOVL    R5,R4                   ; Initialize the loop counter
                                   05EB
                                   05EB ; The comparison is done from most to least significant digits
                                   05EB
                   70   71     D1  05EB 20$:    CMPL    -(R1),-(R0)             ; Compare next pair of digits
                        0E     1F  05EE         BLSSU   30$                     ; Leave loop if product is smaller
                        2D     1A  05F0         BGTRU   50$                     ; Also leave if product is larger
                     F6 54     F4  05F2         SOBGEQ  R4,20$                  ; More to test?
                                   05F5
                                   05F5 ; If we drop through the loop, then the dividend and product are equal. We
                                   05F5 ; simply store zeros in the dividend array (the equivalent of subtraction
                                   05F5 ; of equal arrays) and return. Note that R0 is already pointing to the
                                   05F5 ; least significant dividend array element.
                                   05F5
                   54   55     D0  05F5         MOVL    R5,R4                   ; Initialize still another loop counter
                                   05F8
                        80     D4  05F8 25$:    CLRL    (R0)+                   ; Store another zero
                     FB 54     F4  05FA         SOBGEQ  R4,25$                  ; Keep going?
                                   05FD
                               05  05FD         RSB                             ; Return to caller
                                   05FE
```

```
                          05FE  ; If we drop through the loop, then the quotient that is stored in R3 is good.
                          05FE  ; We need to subtract the product array from the dividend array. Note that R0
                          05FE  ; and R1 need to be adjusted to point to the least significant array elements
                          05FE  ; before the subtraction can begin.
                          05FE
          54   54   CE    05FE  30$:    MNEGL   R4,R4                   ; We need a negative index
       50   6044    DE    0601          MOVAL   (R0)[R4],R0             ; Adjust dividend pointer
       51   6144    DE    0605          MOVAL   (R1)[R4],R1             ; ... and product pointer
          54   55   D0    0609          MOVL    R5,R4                   ; R4 will count still another loop
                          060C
          80   81   C2    060C  35$:    SUBL2   (R1)+,(R0)+             ; Subtract next digits
               0A   18    060F          BGEQ    40$                     ; Skip to end of loop if no borrow
 FC A0   00000064 8F  C0  0611          ADDL2   #100,-4(R0)             ; Add borrow back to this digit
               60   D7    0619          DECL    (R0)                    ; ... and borrow from next highest digit
            EE 54   F4    061B  40$:    SOBGEQ  R4,35$                  ; Keep going?
                          061E
                          061E  ; This is the exit path. R3 contains the quotient digit. The pointers to the
                          061E  ; various input and scratch arrays are in an indeterminate state.
                          061E
                    05    061E  45$:    RSB                             ; Return to caller
                          061F
                          061F  ; The first guess at the quotient digit is too large. The brute force
                          061F  ; approach is to decrement the quotient by one and execute the EMUL/EDIV loop
                          061F  ; again. Note, however, that we can evaluate the modified product by
                          061F  ; subtracting the divisor from the initial product. Note also that, because
                          061F  ; the leading digit in the divisor is "large enough", we can only end up in
                          061F  ; this code path twice. (That is, the initial guess at the quotient will
                          061F  ; never be off by more than two.)
                          061F
               53   D7    061F  50$:    DECL    R3                      ; Try quotient smaller by one
               FB   13    0621          BEQL    45$                     ; All done if zero
                          0623
                          0623  ; Point R1 and R2 at the least significant digits of the scratch and product
                          0623  ; strings respectively.
                          0623
          50   55   CE    0623          MNEGL   R5,R0                   ; Need a negative index
     51   FC A840   DE    0626          MOVAL   -4(R8)[R0],R1           ; Scratch array contains N+1 elements
       52   6740    DE    062B          MOVAL   (R7)[R0],R2             ; Product array contains N elements
          54   55   D0    062F          MOVL    R5,R4                   ; R4 will count still another loop
                          0632
          81   82   C2    0632  60$:    SUBL2   (R2)+,(R1)+             ; Subtract next digits
               0A   18    0635          BGEQ    70$                     ; Skip to end of loop if no borrow
 FC A1   00000064 8F  C0  0637          ADDL2   #100,-4(R1)             ; Add borrow back to this digit
               61   D7    063F          DECL    (R1)                    ; ... and borrow from next highest digit
            EE 54   F4    0641  70$:    SOBGEQ  R4,60$                  ; Keep going?
                          0644
          51   04   C0    0644          ADDL2   #4,R1                   ; Point R1 at most significant digit
               99   11    0647          BRB     15$                     ; Make another comparison
```

```
                                0649        .SUBTITLE       MULTIPLY_STRING - Multiply a String by a Number
                                0649  ;+
                                0649  ; Functional Description:
                                0649  ;
                                0649  ;     This routine multiplies an array of numbers (each array element LEQU
                                0649  ;     99) by a number (also LEQU 99). Each array element in the input array
                                0649  ;     is replaced with the modified product, with the carry propogated to
                                0649  ;     the next array element.
                                0649  ;
                                0649  ; Input Parameters:
                                0649  ;
                                0649  ;     R3 - Multiplier
                                0649  ;     R4 - Input array size
                                0649  ;     R5 - Input array address
                                0649  ;
                                0649  ; Output Parameters:
                                0649  ;
                                0649  ;     None
                                0649  ;
                                0649  ; Implicit Output:
                                0649  ;
                                0649  ;     The input array elements are altered.
                                0649  ;
                                0649  ; Side Effects:
                                0649  ;
                                0649  ;     R4 and R5 are modified by this routine.
                                0649  ;
                                0649  ;     R3 is preserved.
                                0649  ;
                                0649  ;     R0, R1, and R2 are used as scratch registers. R0 and R1 contain the
                                0649  ;     quadword result of EMUL that is then passed into EDIV. R2 is the
                                0649  ;     carry from one step to the next.
                                0649  ;
                                0649  ; Assumptions:
                                0649  ;
                                0649  ;     This routine assumes that all array elements lie in the range from 0
                                0649  ;     to 99 inclusive. (This is true if all input strings contain only legal
                                0649  ;     decimal digits.) The arithmetic performed by this routine will
                                0649  ;     maintain this assumption. The details of this argument can be found in
                                0649  ;     the routine header for EXTENDED_MULTIPLY_STRING. This routine performs
                                0649  ;     less work so that those arguments also apply here.
                                0649  ;-
                                0649
                                0649 MULTIPLY_STRING:
                   52     D4    0649        CLRL    R2                      ; Initial carry is zero
                                064B
       50    52    65  53  7A   064B 10$:   EMUL    R3,(R5),R2,R0           ; Form modified product (R0 LEQU 9900)
 85  52  50  00000064  8F  7B   0650        EDIV    #100,R0,R2,(R5)+        ; Remainder to input array
                                0659                                        ; Quotient becomes carry
                 EF  54    F5   0659        SOBGTR  R4,10$                  ; Keep going?
                                065C
                 65  52    D0   065C        MOVL    R2,(R5)                 ; Store final carry
                           05   065F        RSB
```

```
                   0660          .SUBTITLE       DECIMAL_ROPRAND
                   0660  ;-
                   0660  ; Functional Description:
                   0660  ;
                   0660  ;       This routine receives control when a digit count larger than 31
                   0660  ;       is detected. The exception is architecturally defined as an
                   0660  ;       abort so there is no need to store intermediate state. All of the
                   0660  ;       routines in this module save all registers R0 through R11 before
                   0660  ;       performing the digit check. These registers must be restored
                   0660  ;       before control is passed to VAX$ROPRAND.
                   0660  ;
                   0660  ; Input Parameters:
                   0660  ;
                   0660  ;       00(SP) - Saved R0
                   0660  ;         .
                   0660  ;         .
                   0660  ;       44(SP) - Saved R11
                   0660  ;       48(SP) - Return PC from VAX$xxxxxx routine
                   0660  ;
                   0660  ; Output Parameters:
                   0660  ;
                   0660  ;       00(SP) - Offset in packed register array to delta PC byte
                   0660  ;       04(SP) - Return PC from VAX$xxxxxx routine
                   0660  ;
                   0660  ; Implicit Output:
                   0660  ;
                   0660  ;       This routine passes control to VAX$ROPRAND where further
                   0660  ;       exception processing takes place.
                   0660  ;-
                   0660
                   0660          ASSUME ADDP6_B_DELTA_PC EQ ADDP4_B_DELTA_PC
                   0660          ASSUME SUBP4_B_DELTA_PC EQ ADDP4_B_DELTA_PC
                   0660          ASSUME SUBP6_B_DELTA_PC EQ ADDP4_B_DELTA_PC
                   0660          ASSUME MULP_B_DELTA_PC  EQ ADDP4_B_DELTA_PC
                   0660          ASSUME DIVP_B_DELTA_PC  EQ ADDP4_B_DELTA_PC
                   0660
                   0660  DECIMAL_ROPRAND:
 0FFF 8F   BA      0660          POPR    #^M<R0,R1,R2,R3,R4,R5,R6,R7,R8,R9,R10,R11>
    03     DD      0664          PUSHL   #ADDP4_B_DELTA_PC       ; Store offset to delta PC byte
 F997'     31      0666          BRW     VAX$ROPRAND             ; Pass control along
```

```
                              0669      .SUBTITLE       ARITH_ACCVIO - Reflect an Access Violation
                              0669 ;+
                              0669 ; Functional Description:
                              0669 ;
                              0669 ;        This routine receives control when an access violation occurs while
                              0669 ;        executing within the emulator routines for ADDP4, ADDP6, SUBP4, SUBP6,
                              0669 ;        MULP, or DIVP.
                              0669 ;
                              0669 ;        The routine header for ASHP_ACCVIO in module VAX$ASHP contains a
                              0669 ;        detailed description of access violation handling for the decimal
                              0669 ;        string instructions.
                              0669 ;
                              0669 ; Input Parameters:
                              ^569 ;
                              0669 ;        See routine ASHP_ACCVIO in module VAX$ASHP
                              0669 ;
                              0669 ; Output Parameters:
                              0669 ;
                              0669 ;        See routine ASHP_ACCVIO in module VAX$ASHP
                              0669 ;-
                              0669
                              0669 ARITH_ACCVIO:
                  52    D4    0669      CLRL    R2                      ; Initialize the counter
             F991 CF    9F    066B      PUSHAB  MODULE_BASE             ; Store base address of this module
            06D3'CF     9F    066F      PUSHAB  MODULE_END              ; Store module end address
               F98A'    30    0673      BSBW    DECIMAL$BOUNDS_CHECK    ; Check if PC is inside the module
           5E   04      C0    0676      ADDL    #4,SP                   ; Discard end address
           51   8E      C2    0679      SUBL2   (SP)+,R1                ; Get PC relative to this base
                              067C
      0000'CF42   51    B1    067C 10$:  CMPW    R1,PC_TABLE_BASE[R2]    ; Is this the right PC?
                  07    13    0682      BEQL    30$                     ; Exit loop if true
          F4 52   28    F2    0684      AOBLSS  #TABLE_SIZE,R2,10$      ; Do the entire table
                              0688
                              0688 ; If we drop through the dispatching based on PC, then the exception is not
                              0688 ; one that we want to back up. We simply reflect the exception to the user.
                              0688
                  0F    BA    0688 20$:  POPR    #^M<R0,R1,R2,R3>        ; Restore saved registers
                        05    068A      RSB                             ; Return to exception dispatcher
                              068B
                              068B ; The exception PC matched one of the entries in our PC table. R2 contains
                              068B ; the index into both the PC table and the handler table. R1 has served
                              068B ; its purpose and can be used as a scratch register.
                              068B
  51   0000'CF42        3C    068B 30$:  MOVZWL  HANDLER_TABLE_BASE[R2],R1       ; Get the offset to the handler
       F96A CF41        17    0691      JMP     MODULE_BASE[R1]         ; Pass control to the handler
                              0696
                              0696 ; In all of the instruction-specific routines, the state of the stack
                              0696 ; will be shown as it was when the exception occurred. All offsets will
                              0696 ; be pictured relative to R0.
```

```
                    0696         .SUBTITLE       Access Violation Handling for ADDPx and SUBPx
                    0696  ;+
                    0696  ; Functional Description:
                    0696  ;
                    0696  ;      The only difference among the various entry points is the number of
                    0696  ;      longwords on the stack. R0 is advanced beyond these longwords to point
                    0696  ;      to the list of saved registers. These registers are then restored,
                    0696  ;      effectively backing the routine up to its initial state.
                    0696  ;
                    0696  ; Input Parameters:
                    0696  ;
                    0696  ;      R0 - Address of top of stack when access violation occurred
                    0696  ;
                    0696  ;      See specific entry points for details
                    0696  ;
                    0696  ; Output Parameters:
                    0696  ;
                    0696  ;      See input parameter list for VAX$DECIMAL_ACCVIO in module VAX$ASHP
                    0696  ;-
                    0696
                    0696  ;+
                    0696  ; ADD_SUB_BSBW_24
                    0696  ;
                    0696  ; An access violation occurred in one of the subroutines ADD_PACKED_BYTE,
                    0696  ; SUB_PACKED_BYTE, or STORE_RESULT. In addition to the six longwords of work
                    0696  ; space, this routine has an additional longword, the return PC, on the
                    0696  ; stack.
                    0696  ;
                    0696  ;      00(R0) - Return PC in mainline VAX$xxxxxx routine
                    0696  ;      04(R0) - Address of sign byte of destination string
                    0696  ;      08(R0) - First longword of scratch space
                    0696  ;       etc.
                    0696  ;-
                    0696
                    0696  ADD_SUB_BSBW_24:
 50    04    C0     0696         ADDL    #4,R0                   ; Skip over return PC and drop into ...
                    0699
                    0699  ;+
                    0699  ; ADD_SUB_24
                    0699  ;
                    0699  ; There are five longwords of workspace and a saved string address on the stack
                    0699  ; for this entry point.
                    0699  ;
                    0699  ;      00(R0) - Address of sign byte of destination string
                    0699  ;      04(R0) - First longword of scratch space
                    0699  ;        .
                    0699  ;        .
                    0699  ;      20(R0) - Fifth longword of scratch space
                    0699  ;      24(SP) - Saved R0
                    0699  ;      28(SP) - Saved R1
                    0699  ;       etc.
                    0699  ;-
                    0699
                    0699  ADD_SUB_24:
 50    18    C0     0699         ADDL    #24,R0                  ; Discard scratch space on stack
     F961'   31     069C         BRW     VAX$DECIMAL_ACCVIO      ; Join common code to restore registers
                    069F
```

```
                  069F  ;+
                  069F  ; ADD_SUB_BSBW_0
                  069F  ;
                  069F  ; An access violation occurred in one of the subroutine STRIP_ZEROS. This
                  069F  ; entry point has an additional longword, the return PC, on the stack on top
                  069F  ; of the saved register array.
                  069F  ;
                  069F  ;       00(R0) - Return PC in mainline VAX$xxxxxx routine
                  069F  ;       04(R0) - Saved R0
                  069F  ;       08(R0) - Saved R1
                  069F  ;        etc.
                  069F  ;-
                  069F
                  069F  ADD_SUB_BSBW_0:
50    04    C0    069F          ADDL    #4,R0                   ; Skip over return PC and ...
   F95B'    31    06A2          BRW     VAX$DECIMAL_ACCVIO      ; Join common code to restore registers
```

```
                        06A5         .SUBTITLE       Access Violation Handling for MULP and DIVP
                        06A5  ;+
                        06A5  ; Functional Description:
                        06A5  ;
                        06A5  ;      The only difference among the various entry points is the number of
                        06A5  ;      longwords on the stack. R0 is advanced beyond these longwords to point
                        06A5  ;      to the list of saved registers. These registers are then restored,
                        06A5  ;      effectively backing the routine up to its initial state.
                        06A5  ;
                        06A5  ; Input Parameters:
                        06A5  ;
                        06A5  ;      R0 - Address of top of stack when access violation occurred
                        06A5  ;
                        06A5  ;      See specific entry points for details
                        06A5  ;
                        06A5  ; Output Parameters:
                        06A5  ;
                        06A5  ;      See input parameter list for VAX$DECIMAL_ACCVIO in module VAX$ASHP
                        06A5  ;-
                        06A5
                        06A5  ;+
                        06A5  ; MULP_R8
                        06A5  ;
                        06A5  ; An access violation occurred while MULP was accessing one of its two source
                        06A5  ; strings. In this particular case, MULP was storing the longer of the two
                        06A5  ; input strings in a longword array on the top of the stack. There is an
                        06A5  ; array of R8 longwords on top of an array of 32 longwords on top of the
                        06A5  ; saved register array.
                        06A5  ;
                        06A5  ;      R8 - Number of longwords on top of the 32-longword array
                        06A5  ;-
                        06A5
                        06A5         .ENABLE         LOCAL_BLOCK
                        06A5
                        06A5  MULP_R8:
   50    6048     DE    06A5         MOVAL   (R0)[R8],R0             ; Discard input array storage
           04     11    06A9         BRB     10$                     ; Might as well share a little code
                        06AB
                        06AB  ;+
                        06AB  ; MULP_AT_SP
                        06AB  ;
                        06AB  ; An access violation occurred while MULP was accessing one of its two source
                        06AB  ; strings. In this case, the access violation occurred in the middle of the
                        06AB  ; grand multiply loop as a digit pair was being retrieved from the shorter of
                        06AB  ; the two input strings. The address of the start of the 32-longword array
                        06AB  ; was itself stored on top of the stack for convenience.
                        06AB  ;
                        06AB  ;      00(R0) - Saved byte count of longer input string
                        06AB  ;      04(R0) - Saved address of longer input string
                        06AB  ;      08(R0) - Address of 32-longword array farther down the stack
                        06AB  ;-
                        06AB
                        06AB  MULP_AT_SP:
   50   08 A0     D0    06AB         MOVL    8(R0),R0                ; Locate start of 32-longword array
  50  0088 C0     9E    06AF  10$:   MOVAB   <<4*32> + <4*2>>(R0),R0 ; Throw that away, too
        F949'     31    06B4         BRW     VAX$DECIMAL_ACCVIO      ; Join common code to restore registers
                        06B7
```

```
                        06B7          .DISABLE        LOCAL_BLOCK
                        06B7
                        06B7     ;+
                        06B7     ; MULP_DIVP_R9
                        06B7     ;
                        06B7     ; An access violation occurred while the final result was being stored in the
                        06B7     ; result string. In this common exit code path, R9 counts the number of
                        06B7     ; longwords on the stack. In all cases where an access violation can occur, a
                        06B7     ; longword has been removed from the stack but R9 has not yet been
                        06B7     ; decremented to reflect this. The conceptual instruction sequence that
                        06B7     ; resets the stack pointer (really R0) to point to the start of the saved
                        06B7     ; register array is
                        06B7     ;
                        06B7     ;       DECL    R9
                        06B7     ;       MOVAL   (R0)[R9]
                        06B7     ;
                        06B7     ; A single instruction accomplishes this.
                        06B7     ;
                        06B7     ;       R9 - One more than the number of longwords on the stack on top
                        06B7     ;               of the saved register array.
                        06B7     ;
                        06B7     ;       00(R0) - First longword of scratch storage remaining on the stack
                        06B7     ;         .
                        06B7     ;         .
                        06B7     ;       zz-4(R0)  - Last longword of scratch storage
                        06B7     ;       zz+0(R0)  - Saved count of dividend or multiplier string
                        06B7     ;       zz+4(R0)  - Saved address of dividend or multiplier string
                        06B7     ;       zz+8(R0)  - Saved R0
                        06B7     ;       zz+12(R0) - Saved R1
                        06B7     ;        etc.
                        06B7     ;
                        06B7     ;       where zz = 4 * (R9 - 1)
                        06B7     ;-
                        06B7
                        06B7     MULP_DIVP_R9:
   50   04 A049   DE    06B7          MOVAL   4(R0)[R9],R0          ; Discard scratch storage on stack
          F941'   31    06BC          BRW     VAX$DECIMAL_ACCVIO    ; Join common code to restore registers
                        06BF
                        06BF     ;+
                        06BF     ; MULP_DIVP_8
                        06BF     ;
                        06BF     ; An access violation occurred in the common exit path after the scratch array
                        06BF     ; had been removed from the stack but before the saved descriptor for the
                        06BF     ; multiplier string was discarded.
                        06BF     ;
                        06BF     ;       0(R0)  - Saved count of dividend or multiplier string
                        06BF     ;       4(R0)  - Saved address of dividend or multiplier string
                        06BF     ;       8(R0)  - Saved R0
                        06BF     ;       12(R0) - Saved R1
                        06BF     ;        etc.
                        06BF     ;-
                        06BF
                        06BF     MULP_DIVP_8:
        50   08   C0    06BF          ADDL    #8,R0                 ; Discard multiplier string descriptor
          F93B'   31    06C2          BRW     VAX$DECIMAL_ACCVIO    ; Join common code to restore registers
                        06C5
                        06C5     ;+
```

```
                    06C5  ; MULP_BSBW_0
                    06C5  ; DIVP_BSBW_0
                    06C5  ;
                    06C5  ; An access violation occurred in one of the subroutine STRIP_ZEROS. This
                    06C5  ; entry point has an additional longword, the return PC, on the stack on top
                    06C5  ; of the saved register array.
                    06C5  ;
                    06C5  ;       00(R0) - Return PC in mainline VAX$MULP or VAX$DIVP routine
                    06C5  ;       04(R0) - Saved R0
                    06C5  ;       08(R0) - Saved R1
                    06C5  ;        etc.
                    06C5  ;-
                    06C5
                    06C5  MULP_BSBW_0:
                    06C5  DIVP_BSBW_0:
 50     04     C0   06C5          ADDL    #4,R0                   ; Skip over return PC and drop into ...
                    06C8
                    06C8  ;+
                    06C8  ; DIVP_0
                    06C8  ; MULP_DIVP_0
                    06C8  ;
                    06C8  ; There was nothing allocated on the stack other than the saved register
                    06C8  ; array when the access violation occurred. We merely pass control to common
                    06C8  ; code to restore the registers.
                    06C8  ;
                    06C8  ;       00(R0) - Saved R0
                    06C8  ;       04(R0) - Saved R1
                    06C8  ;        etc.
                    06C8  ;-
                    06C8
                    06C8  DIVP_0:
                    06C8  MULP_DIVP_0:
      F935'    31   06C8          BRW     VAX$DECIMAL_ACCVIO      ; Join common code to restore registers
                    06CB
                    06CB  ;+
                    06CB  ; DIVP_R6_R7
                    06CB  ;
                    06CB  ; An access violation occurred while one of the two input strings was being
                    06CB  ; converted to an array of longwords on the stack. The state of the stack
                    06CB  ; is rather complicated but R6 and R7 contain enough information to allow
                    06CB  ; the rest of the stack contents to be ignored.
                    06CB  ;
                    06CB  ;       R6 - Count of longwords in quotient array on stack
                    06CB  ;       R7 - Address of quotient array on stack
                    06CB  ;
                    06CB  ;       00(R0) - First longword of quotient array
                    06CB  ;         .
                    06CB  ;         .
                    06CB  ;       zz-4(R0) - Last longword of scratch storage
                    06CB  ;       zz+0(R0) - Digit count of dividend string
                    06CB  ;       zz+4(R0) - Address of dividend string
                    06CB  ;       zz+8(R0) - Saved R0
                    06CB  ;       zz+12(R0) - Saved R1
                    06CB  ;        etc.
                    06CB  ;
                    06CB  ;       where zz = 4 * R6
                    06CB  ;-
```

```
                         06CB  2490
                         06CB  2491 DIVP_R6_R7:
       50   08 A746   DE 06CB  2492         MOVAL   8(R7)[R6],R0            ; Discard everything on stack
                F92D' 31 06D0  2493         BRW     VAX$DECIMAL_ACCVIO      ; Join common code to restore registers
                         06D3  2494
                         06D3  2495         END_MARK_POINT
                         06D3  2496
                         06D3  2497         .END
```

```
...PC...                          = 0000052D
...ROPRAND...                     = 00000499 R      02
ADDP4_B_DELTA_PC                  = 00000003
ADDP6_B_DELTA_PC                  = 00000003
ADD_PACKED                          000000DA R      02
ADD_PACKED_BYTE_R6_R7               00000165 R      02
ADD_PACKED_BYTE_STRING              0000015F R      02
ADD_SUBTRACT_EXIT                   00000132 R      02
ADD_SUB_24                          00000699 R      02
ADD_SUB_BSBW_0                      0000069F R      02
ADD_SUB_BSBW_24                     00000696 R      02
ADD_SUB_V_ZERO_R4                 = 0000001F
ARITH_ACCVIO                        00000669 R      02
DECIMAL$BINARY_TO_PACKED_TABLE      ********   X    00
DECIMAL$BOUNDS_CHECK                ********   X    00
DECIMAL$PACKED_TO_BINARY_TABLE      ********   X    00
DECIMAL$STRIP_ZEROS_R0_R1           ********   X    00
DECIMAL$STRIP_ZEROS_R2_R3           ********   X    00
DECIMAL_ROPRAND                     00000660 R      02
DIVIDE_BY_ZERO                      00000478 R      02
DIVP_0                              000006C8 R      02
DIVP_BSBW_0                         000006C5 R      02
DIVP_B_DELTA_PC                   = 00000003
DIVP_R6_R7                          000006CB R      02
EXTEND_STRING_MULTIPLY              0000044A R      02
HANDLER_TABLE_BASE                  00000000 R      04
MODULE_BASE                       = 00000000 R      02
MODULE_END                        = 000006D3 R      02
MULP_AT_SP                          000006AB R      02
MULP_BSBW_0                         000006C5 R      02
MULP_B_DELTA_PC                   = 00000003
MULP_DIVP_0                         000006C8 R      02
MULP_DIVP_8                         000006BF R      02
MULP_DIVP_R9                        000006B7 R      02
MULP_R8                             000006A5 R      02
MULTIPLY_DIVIDE_EXIT                00000357 R      02
MULTIPLY_STRING                     00000649 R      02
PC_TABLE_BASE                       00000000 R      03
PSL$M_N                           = 00000008
PSL$M_V                           = 00000002
PSL$M_Z                           = 00000004
PSL$V_CM                          = 0000001F
PSL$V_V                           = 00000001
PSL$V_Z                           = 00000002
QUOTIENT_DIGIT                      000005A4 R      02
SRM$K_FLT_DIV_T                   = 00000004
STORE_RESULT                        00000249 R      02
SUBP4_B_DELTA_PC                  = 00000003
SUBP6_B_DELTA_PC                  = 00000003
SUBTRACT_PACKED                     0000018D R      02
SUB_PACKED_BYTE_R6_R7               00000223 R      02
SUB_PACKED_BYTE_STRING              0000021D R      02
TABLE_SIZE                        = 00000028
VAX$ADDP4                           0000002B RG     02
VAX$ADDP6                           00000009 RG     02
VAX$ADD_PACKED_BYTE_R6_R7           00000165 RG     02
VAX$DECIMAL_ACCVIO                  ********   X    00
VAX$DECIMAL_EXIT                    ********   X    00
VAX$DIVP                            0000048B RG     02
VAX$MULP                            00000287 RG     02
VAX$REFLECT_TRAP                    ********   X    00
VAX$ROPRAND                         ********   X    00
VAX$SUBP4                           00000022 RG     02
VAX$SUBP6                           00000000 RG     02
```

```
                                                       +----------------+
                                                       ! 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                       000006D3  (  1747.)  02 (  2.)    PIC   USR   CON   REL   LCL   SHR   EXE   RD   NOWRT NOVEC LONG
PC_TABLE                        00000050  (    80.)  03 (  3.)    PIC   USR   CON   REL   LCL   SHR NOEXE   RD   NOWRT NOVEC BYTE
HANDLER_TABLE                   00000050  (    80.)  04 (  4.)    PIC   USR   CON   REL   LCL   SHR NOEXE   RD   NOWRT NOVEC BYTE

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

Phase                     Page faults    CPU Time       Elapsed Time
-----                     -----------    --------       ------------
Initialization                    10     00:00:00.06    00:00:00.99
Command processing                71     00:00:00.55    00:00:03.24
Pass 1                           208     00:00:07.77    00:00:22.36
Symbol table sort                  0     00:00:00.35    00:00:01.58
Pass 2                           392     00:00:04.76    00:00:13.45
Symbol table output                0     00:00:00.06    00:00:00.62
Psect synopsis output              0     00:00:00.03    00:00:00.03
Cross-reference output             0     00:00:00.00    00:00:00.00
Assembler run totals             681     00:00:13.58    00:00:42.27

The working set limit was 1650 pages.
50323 bytes (99 pages) of virtual memory were used to buffer the intermediate code.
There were 20 pages of symbol table space allocated to hold 182 non-local and 113 local symbols.
2497 source lines were read in Pass 1, producing 25 object records in Pass 2.
23 pages of virtual memory were used to define 21 macros.

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

Macro library name                                 Macros defined
------------------                                 --------------
_$255$DUA28:[EMULAT.OBJ]VAXMACROS.MLB;1                  12
_$255$DUA28:[SYSLIB]STARLET.MLB;2                         6
TOTALS (all libraries)                                   18

318 GETS were required to define 18 macros.

There were no errors, warnings or information messages.

MACRO/LIS=LIS$:VAXARITH/OBJ=OBJ$:VAXARITH MSRC$:VAXARITH/UPDATE=(ENH$:VAXARITH)+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