D 6

```
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

VAXARITH.

LIS

VAX
VO4

Table of contents

```
          0000      .TITLE  VAX$DECIMAL_ARITHMETIC - VAX-11 Packed Decimal Arithmetic Instructio
;LJK0045  0000      .IDENT  /V04-001/
-1        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  ;
;LJK0045            0000  ;       V04-001 LJK0045         Lawrence J. Kenah       19-Sep-1984
;LJK0045            0000  ;               The result string in ADDP4 and SUBP4 must be probed for
;LJK0045            0000  ;               write access to insure that the instruction does not modify
;LJK0045            0000  ;               memory before trying to restart.
;LJK0045            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
```

```
-1
                     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$:    ROPRAND_CHECK   R4                      ; Insure that R4 is LEQU 31
               5B    DC  001A          MOVPSL  R11                             ; Get initial PSL
                         001C
                         001C  ; Indicate that the saved R4 must be cleared on the exit path
                         001C
;LJK0045  22 5B  1F  E3  001C          BBCS    #ADD_SUB_V_ZERO_R4,R11,25$      ; Set bit and join common code
;LJK0045         20  11  0020          BRB     25$                             ; 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
     0B  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   ; Condition 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
;LJK0045           0264     30     002B   15$:    BSBW    CHECK_WRITE_ACCESS      ; Perform rigorous access check
;LJK0045             38     11     002E           BRB     30$                     ; String can be written after all
;LJK0045                           0030
                                   0030   VAX$ADDP4::
                OFFF 8F     BB     0030           PUSHR   #^M<R0,R1,R2,R3,R4,R5,R6,R7,R8,R9,R10,R11>      ; Save the lot
                     59     D4     0034           CLRL    R9                      ; This is addition
                                   0036
                                   0036   ; The output string, described by R4 and R5, will be the same as the input
                                   0036   ; string for ADDP4 and SUBP4. It is necessary to explicitly clear R4<31:16>
                                   0036   ; along this code path so MOVQ R2,R4 will not always work.
                                   0036
                  54 52     3C     0036   20$:    MOVZWL  R2,R4                   ; Set output size equal to input size
                  55 53     D0     0039           MOVL    R3,R5                   ; ... and ditto for string addresses
                     5B     DC     003C           MOVPSL  R11                     ; Get initial PSL
                                   003E
                                   003E   ; Indicate that the saved R4 will be restored on the common exit path
                                   003E
;LJK0045        00 5B 1F    E5     003E           BBCC    #ADD_SUB_V_ZERO_R4,R11,25$      ; Clear bit and join common code
```

```
                                    0042              .SUBTITLE        ADDPx/SUBPx Common Initialization Code
                                    0042        ;+
                                    0042        ; All four routines converge at this point and execute common initialization
                                    0042        ; code until a later decision is made to do addition or subtraction.
                                    0042        ;
                                    0042        ;       R4 - Number of digits in destination string
                                    0042        ;       R5 - Address of destination string
                                    0042        ;
                                    0042        ;       R9 - Indicates whether operation is addition or subtraction
                                    0042        ;               0 => addition
                                    0042        ;               1 => subtraction
                                    0042        ;
                                    0042        ;       R11<31> - Indicates whether this is a 4-operand or 6-operand instruction
                                    0042        ;               0 => 4-operand (restore saved R4 on exit)
                                    0042        ;               1 => 6-operand (set R4 to zero on exit)
                                    0042        ;-
                                    0042
;LJK0045          04   00   04   F0  0042   25$:   INSV    #PSL$M_Z,#0,#4,R11      ; Set Z-bit, clear the rest in saved PSW
                            5B      0046
-1                                  0047          ESTABLISH_HANDLER        -      ; Store address of access
                                    0047                  ARITH_ACCVIO            ;  violation handler
                                    004C
                                    004C          ROPRAND_CHECK   R2                      ; Insure that R2 is LEQU 31
                                    0054          MARK_POINT      ADD_SUB_BSBW_0
                        FFA9'  30   0054          BSBW    DECIMAL$STRIP_ZEROS_R2_R3       ; Strip high order zeros from R2/R3
                                    0057
                                    0057          ROPRAND_CHECK   R0                      ; Insure that R0 is LEQU 31
                                    005F          MARK_POINT      ADD_SUB_BSBW_0
                        FF9E'  30   005F          BSBW    DECIMAL$STRIP_ZEROS_R0_R1       ; Strip high order zeros from R0/R1
                                    0062
;LJK0045                            0062   ; Perform the access check on the output string for the worst case, a string
;LJK0045                            0062   ; large enough to accommodate 31 decimal digits. A detailed check using the
;LJK0045                            0062   ; correct byte length of the output string is necessary only if this initial
;LJK0045                            0062   ; probe fails. The more detailed check can be handled out of line.
;LJK0045                            0062
;LJK0045          65   10   00   0D  0062          PROBEW  #0,#16,(R5)             ; Can result string be written?
;LJK0045                    C3   13  0066          BEQL    15$                     ; Branch if no write access allowed
;LJK0045                            0068
                                    0068   ; Rather than totally confuse the already complicated logic dealing with
                                    0068   ; different length strings in the add or subtract loop, we will put the
                                    0068   ; result into an intermediate buffer on the stack. This buffer will be long
                                    0068   ; enough to handle the worst case so that the addition loop need only concern
                                    0068   ; itself with the lengths of the two input loops. The required length is 17
                                    0068   ; bytes, to handle an addition with a carry out of the most significant byte.
                                    0068   ; We will allocate 20 bytes to maintain whatever alignment the stack has.
                                    0068
;LJK0045                    7E   7C  0068   30$:   CLRQ    -(SP)                   ; Set aside space for output string
-1                          7E   7C  006A          CLRQ    -(SP)                   ; Worst case string needs 16 bytes
                            7E   D4  006C          CLRL    -(SP)                   ; Add slack for a CARRY
                  54   04   01   EF  006E          EXTZV   #1,#4,R4,R8             ; Get byte count of destination string
                            58      0072
                  7E   55   58   C1  0073          ADDL3   R8,R5,-(SP)             ; Save high address end of destination
                     55   18 AE   9E  0077          MOVAB   24(SP),R5               ; Point R5 one byte beyond buffer
                                    007B
                                    007B   ; The number of minus signs will determine whether the real operation that we
                                    007B   ; perform is addition or subtraction. That is, two plus signs or two minus
                                    007B   ; signs will both result in addition, while a plus sign and a minus sign will
```

```
                    007B   ; result in subtraction. The addition and subtraction routines have their own
                    007B   ; methods for determining the correct sign of the result.
                    007B   ;
                    007B   ; For the purpose of counting minus signs, we treat subtraction as the
                    007B   ; addition of the negative of the input operand. That is, subtraction of a
                    007B   ; positive quantity causes the sign to be remembered as minus and counted as
                    007B   ; a minus sign while subtraction of a minus quantity stores a plus sign and
                    007B   ; counts nothing.
                    007B   ;
                    007B   ; On input to this code sequence, R9 distinguished addition from subtraction.
                    007B   ; On output, it contains either 0, 1, or 2, indicating the total number of
                    007B   ; minus signs, real or implied, that we counted.
                    007B
50    04    01  EF  007B            EXTZV   #1,#4,R0,R6               ; Get byte count for first input string
            56      007F
      51    56  C0  0080            ADDL    R6,R1                     ; Point R1 to byte containing sign
                    0083            MARK_POINT      ADD_SUB_24
   61    F0 8F  8B  0083            BICB3   #^B11110000,(R1),R6       ; R6 contains the sign "digit"
            56      0087
      10    59  E8  0088            BLBS    R9,35$                    ; Use second CASE if subtraction
                    008B
                    008B   ; This case statement is used for addition
                    008B
                    008B            CASE    R6,TYPE=B,LIMIT=#10,<-    ; Dispatch on sign digit
                    008B                    50$,-                     ; 10 => sign is "+"
                    008B                    40$,-                     ; 11 => sign is "-"
                    008B                    50$,-                     ; 12 => sign is "+"
                    008B                    40$,-                     ; 13 => sign is "-"
                    008B                    50$,-                     ; 14 => sign is "+"
                    008B                    50$,-                     ; 15 => sign is "+"
                    008B                    >
                    009B
                    009B   ; This case statement is used for subtraction
                    009B
                    009B   35$:     CASE    R6,TYPE=B,LIMIT=#10,<-    ; Dispatch on sign digit
                    009B                    40$,-                     ; 10 => treat sign as "-"
                    009B                    50$,-                     ; 11 => treat sign as "+"
                    009B                    40$,-                     ; 12 => treat sign as "-"
                    009B                    50$,-                     ; 13 => treat sign as "+"
                    009B                    40$,-                     ; 14 => treat sign as "-"
                    009B                    40$,-                     ; 15 => treat sign as "-"
                    009B                    >
                    00AB
      59    01  D0  00AB   40$:     MOVL    #1,R9                     ; Count a minus sign
      56    0D  9A  00AE            MOVZBL  #13,R6                    ; The preferred minus sign is 13
            05  11  00B1            BRB     60$                       ; Now check second input sign
                    00B3
            59  D4  00B3   50$:     CLRL    R9                        ; No real minus signs so far
      56    0C  9A  00B5            MOVZBL  #12,R6                    ; The preferred minus sign is 12
                    00B8
52    04    01  EF  00B8   60$:     EXTZV   #1,#4,R2,R7               ; Get byte count for second input string
            57      00BC
      53    57  C0  00BD            ADDL    R7,R3                     ; Point R3 to byte containing sign
                    00C0            MARK_POINT      ADD_SUB_24
   63    F0 8F  8B  00C0            BICB3   #^B11110000,(R3),R7       ; R7 contains the sign "digit"
            57      00C4
                    00C5
```

```
                   00C5            CASE    R7,TYPE=B,LIMIT=#10,<-   ; Dispatch on sign digit
                   00C5                    80$,-                    ; 10 => sign is "+"
                   00C5                    70$,-                    ; 11 => sign is "-"
                   00C5                    80$,-                    ; 12 => sign is "+"
                   00C5                    70$,-                    ; 13 => sign is "-"
                   00C5                    80$,-                    ; 14 => sign is "+"
                   00C5                    80$,-                    ; 15 => sign is "+"
                   00C5                    >
                   00D5
         59   D6   00D5    70$:    INCL    R9                       ; Remember that sign was minus
      57 0D   9A   00D7            MOVZBL  #13,R7                   ; The preferred minus sign is 13
         03   11   00DA            BRB     90$                      ; Now check second input sign
                   00DC
      57 0C   9A   00DC    80$:    MOVZBL  #12,R7                   ; The preferred minus sign is 12
                   00DF
      03 59   E9   00DF    90$:    BLBC    R9,ADD_PACKED            ; Even parity indicates addition
                   00E2
       00B3   31   00E2            BRW     SUBTRACT_PACKED          ; Odd parity calls for subtraction
                   00E5
                   00E5            .DISABLE        LOCAL_BLOCK
```

```
00E5        .SUBTITLE       ADD_PACKED - Add Two Packed Decimal Strings
00E5  ;+
00E5  ; Functional Description:
00E5  ;
00E5  ;       This routine adds two packed decimal strings whose descriptors
00E5  ;       are passed as input parameters and places their sum into another
00E5  ;       (perhaps identical) packed decimal string.
00E5  ;
00E5  ;       At the present time, the result is placed into a 16-byte storage
00E5  ;       area while the sum is being evaluated. This drastically reduces
00E5  ;       the number of different cases that must be dealt with as each
00E5  ;       pair of bytes in the two input strings is added.
00E5  ;
00E5  ;       The signs of the two input strings have already been dealt with
00E5  ;       so this routine performs addition in all cases, even if the original
00E5  ;       entry was at SUBP4 or SUBP6. The cases that arrive in this routine
00E5  ;       are as follows.
00E5  ;
00E5  ;                             R2/R3          R0/R1          result
00E5  ;                       +--------------+--------------+--------------+
00E5  ;                       :              :              :              :
00E5  ;       R2/R3 + R0/R1   :     plus     :     plus     :     plus     :
00E5  ;                       :              :              :              :
00E5  ;                       +--------------+--------------+--------------+
00E5  ;                       :              :              :              :
00E5  ;       R2/R3 + R0/R1   :    minus     :    minus     :    minus     :
00E5  ;                       :              :              :              :
00E5  ;                       +--------------+--------------+--------------+
00E5  ;                       :              :              :              :
00E5  ;       R2/R3 - R0/R1   :    minus     :     plus     :    minus     :
00E5  ;                       :              :              :              :
00E5  ;                       +--------------+--------------+--------------+
00E5  ;                       :              :              :              :
00E5  ;       R2/R3 - R0/R1   :     plus     :    minus     :     plus     :
00E5  ;                       :              :              :              :
00E5  ;                       +--------------+--------------+--------------+
00E5  ;
00E5  ;       Note that the correct choice of sign in all four cases is the sign
00E5  ;       of the second input string, the one described by R2 and R3.
00E5  ;
00E5  ; Input Parameters:
00E5  ;
00E5  ;       R0<4:0> - Number of digits in first input decimal string
00E5  ;       R1      - Address of least significant digit of first input
00E5  ;                 decimal string (the byte containing the sign)
00E5  ;
00E5  ;       R2<4:0> - Number of digits in second input decimal string
00E5  ;       R3      - Address of least significant digit of second input
00E5  ;                 decimal string (the byte containing the sign)
00E5  ;
00E5  ;       R4<4:0> - Number of digits in output decimal string
00E5  ;       R5      - Address of one byte beyond least significant digit of
00E5  ;                 intermediate string stored on the stack
00E5  ;
00E5  ;       R6<3:0> - Sign of first input string in preferred form
00E5  ;       R7<3:0> - Sign of second input string in preferred form
00E5  ;
```

```
                     00E5  ;       R11      - Saved PSL (Z-bit is set, other condition codes are clear)
                     00E5  ;
                     00E5  ;       (SP)     - Saved R5, address of least significant digit of ultimate
                     00E5  ;                  destination string.
                     00E5  ;       4(SP)    - Beginning of 20-byte buffer to hold intermediate result
                     00E5  ;
                     00E5  ; Output Parameters:
                     00E5  ;
                     00E5  ;       The particular input operation (ADDPx or SUBPx) is completed in
                     00E5  ;       this routine. See the routine headers for the four routines that
                     00E5  ;       request addition or subtraction for a list of output parameters
                     00E5  ;       from this routine.
                     00E5  ;-
                     00E5
                     00E5  ADD_PACKED:
       59   57   90  00E5          MOVB    R7,R9                    ; Use sign of second string for output
        03 59    E9  00E8          BLBC    R9,10$                   ; Check if sign is negative
       5B   08   88  00EB          BISB    #PSL$M_N,R11             ; ... so the saved N-bit can be set
                     00EE
                     00EE          MARK_POINT      ADD_SUB_24
  56   61   0F   8B  00EE  10$:    BICB3   #^B00001111,(R1),R6      ; Get least significant digit to R6
                     00F2          MARK_POINT      ADD_SUB_24
  57   63   0F   8B  00F2          BICB3   #^B00001111,(R3),R7      ; Get least significant digit to R7
            58   D4  00F6          CLRL    R8                       ; Start the add with CARRY off
        0075     30  00F8          BSBW    ADD_PACKED_BYTE_R6_R7    ; Add the two low order digits
                     00FB
                     00FB  ; The following set of instructions computes the number of bytes in the two
                     00FB  ; strings and, if necessary, performs a switch so that R0 and R1 always
                     00FB  ; describe the shorter of the two strings.
                     00FB
  50   04   01   EF  00FB          EXTZV   #1,#4,R0,R0              ; Convert digit count to byte count
            50       00FF
  52   04   01   EF  0100          EXTZV   #1,#4,R2,R2              ; Do it for both strings
            52       0104
       52   50   D1  0105          CMPL    R0,R2                    ; We want to compare the byte counts
            09   1B  0108          BLEQU   20$                      ; Skip the swap if we're already correct
       56   50   7D  010A          MOVQ    R0,R6                    ; Save the longer
       50   52   7D  010D          MOVQ    R2,R0                    ; Store the shorter on R0 and R1
       52   56   7D  0110          MOVQ    R6,R2                    ; ... and store the longer in R2 and R3
       52   50   C2  0113  20$:    SUBL    R0,R2                    ; Make R2 a difference (R2 GEQU 0)
                     0116
                     0116  ; R0 now contains the number of bytes remaining in the shorter string.
                     0116  ; R2 contains the difference in bytes between the two input strings.
                     0116
            50   D5  0116          TSTL    R0                       ; Does shorter string have any room?
            06   13  0118          BEQL    40$                      ; Skip loop if no room at all
                     011A
        004D     30  011A  30$:    BSBW    ADD_PACKED_BYTE_STRING   ; Add the next two bytes together
        FA 50    F5  011D          SOBGTR  R0,30$                   ; Check for end of loop
                     0120
            52   D5  0120  40$:    TSTL    R2                       ; Does longer string have any room?
            16   13  0122          BEQL    70$                      ; Skip next loops if all done
                     0124
        0D 58    E9  0124  50$:    BLBC    R8,60$                   ; Life is simple if CARRY clear
                     0127
            56   D4  0127          CLRL    R6                       ; Otherwise, CARRY must propogate
                     0129          MARK_POINT      ADD_SUB_24
```

```
      57   73  9A 0129  569          MOVZBL  -(R3),R7                     ; So add CARRY to single string
         0041  30 012C  570          BSBW    ADD_PACKED_BYTE_R6_R7        ; Use the special entry point
       F2  52  F5 012F  571          SOBGTR  R2,50$                       ; Check for this string exhausted
           06  11 0132  572          BRB     70$                          ; Join common completion code
              0134  573
              0134  574          MARK_POINT      ADD_SUB_24
      75   73  90 0134  575 60$:     MOVB    -(R3),-(R5)                  ; Simply move src to dst if no CARRY
       FA  52  F5 0137  576          SOBGTR  R2,60$                       ; ... until we're all done
              013A  577
      75   58  90 013A  578 70$:     MOVB    R8,-(R5)                     ; Store the final CARRY
              013D  579
              013D  580 ;+
              013D  581 ; At this point, the result has been computed. That result must be moved to
              013D  582 ; its ultimate destination, noting whether any nonzero digits are stored
              013D  583 ; so that the Z-bit will have its correct setting.
              013D  584 ;
              013D  585 ; Input Parameters:
              013D  586 ;
              013D  587 ;       R9<7:0>  - Sign of result in preferred form
              013D  588 ;       R11<3:0> - Saved condition codes
              013D  589 ;       R11<31>  - Indicates whether to set saved R4 to zero
              013D  590 ;
              013D  591 ;       (SP)     - Saved R5, high address end of destination string
              013D  592 ;-
              013D  593
              013D  594 ADD_SUBTRACT_EXIT:
   55  6E  01  C1 013D  595          ADDL3   #1,(SP),R5                   ; Point R5 beyond real destination
   51   18 AE  9E 0141  596          MOVAB   24(SP),R1                    ; R1 locates the saved result
         010C  30 0145  597          BSBW    STORE_RESULT                 ; Store the result and record the Z-bit
   12  5B  02  E0 0148  598          BBS     #PSL$V_Z,R11,100$            ; Step out of line for minus zero check
              014C  599
              014C  600          MARK_POINT      ADD_SUB_24
   04  00  59  F0 014C  601 80$:     INSV    R9,#0,#4,a(SP)+              ; The sign can finally be stored
           9E     0150
           14  C0 0151  602          ADDL    #20,SP                       ; Get rid of intermediate buffer
   03  5B  1F  E1 0154  603          BBC     #ADD_SUB_V_ZERO_R4,R11,90$        ; Branch if 4-operand opcode
        10 AE  D4 0158  604          CLRL    16(SP)                       ; Clear saved R4 to return zero
        FEA2'  31 015B  605 90$:     BRW     VAX$DECIMAL_EXIT             ; Exit through common code path
              015E  606
              015E  607 ; If the result is negative zero, then the N-bit is cleared and the sign
              015E  608 ; is changed to a plus sign.
              015E  609
       5B  08  8A 015E  610 100$:    BICB    #PSL$M_N,R11                 ; Clear the N-bit unconditionally
   E7  5B  01  E0 0161  611          BBS     #PSL$V_V,R11,80$             ; Do not change the sign on overflow
       59  0C  90 0165  612          MOVB    #12,R9                       ; Make sure that the sign is plus
           E2  11 0168  613          BRB     80$                          ; ... and rejoin the exit code
```

```
016A        .SUBTITLE       ADD_PACKED_BYTE - Add Two Bytes Containing Decimal Digits
016A ;+
016A ; Functional Description:
016A ;
016A ;      This routine adds together two bytes containing decimal digits and
016A ;      produces a byte containing the sum that is stored in the output
016A ;      string. Each of the input bytes is converted to a binary number
016A ;      (with a table-driven conversion), the two numbers are added, and
016A ;      the sum is converted back to two decimal digits stored in a byte.
016A ;
016A ;      This routine makes no provisions for bytes that contain illegal
016A ;      decimal digits. We are using the UNPREDICTABLE statement in the
016A ;      architectural description of the decimal instructions to its fullest.
016A ;
016A ;      The bytes that contain a pair of packed decimal digits can either
016A ;      exist in packed decimal strings located by R1 and R3 or they can
016A ;      be stored directly in registers. In the former case, the digits must
016A ;      be extracted from registers before they can be used in later operations
016A ;      because the sum will be used as an index register.
016A ;
016A ; For entry at ADD_PACKED_BYTE_STRING:
016A ;
016A ;      Input Parameters:
016A ;
016A ;              R1  - Address one byte beyond first byte that is to be added
016A ;              R3  - Address one byte beyond second byte that is to be added
016A ;              R5  - Address one byte beyond location to store sum
016A ;
016A ;              R8  - Carry from previous byte (R8 is either 0 or 1)
016A ;
016A ;      Implicit Input:
016A ;
016A ;              R6 - Scratch
016A ;              R7 - Scratch
016A ;
016A ;      Output Parameters:
016A ;
016A ;              R1 - Decreased by one to point to current byte in first input string
016A ;              R3 - Decreased by one to point to current byte in second input strin
016A ;              R5 - Decreased by one to point to current byte in output string
016A ;
016A ;              R8 - Either 0 or 1, reflecting whether this most recent ADD resulted
016A ;                   in a CARRY to the next byte.
016A ;
016A ; For entry at ADD_PACKED_BYTE_R6_R7:
016A ;
016A ;      Input Parameters:
016A ;
016A ;              R6  - First byte containing decimal digit pair
016A ;              R7  - Second byte containing decimal digit pair
016A ;
016A ;              R5  - Address one byte beyond location to store sum
016A ;
016A ;              R8  - Carry from previous byte (R8 is either 0 or 1)
016A ;
016A ;      Output Parameters:
016A ;
```

```
                          016A  ;              R5 - Decreased by one to point to current byte in output string
                          016A  ;
                          016A  ;              R8 - Either 0 or 1, reflecting whether this most recent ADD resulted
                          016A  ;                   in a CARRY to the next byte.
                          016A  ;
                          016A  ; Side Effects:
                          016A  ;
                          016A  ;       R6 and R7 are modified by this routine
                          016A  ;
                          016A  ;       R0, R2, R4, and R9 (and, of course, R10 and R11) are preserved
                          016A  ;       by this routine
                          016A  ;
                          016A  ; Assumptions:
                          016A  ;
                          016A  ;       This routine makes two important assumptions.
                          016A  ;
                          016A  ;       1.  If both of the input bytes contain only legal decimal digits, then
                          016A  ;           it is only necessary to subtract 100 at most once to put all
                          016A  ;           possible sums in the range 0..99. That is,
                          016A  ;
                          016A  ;               99 + 99 + 1 = 199 LSS 200
                          016A  ;
                          016A  ;       2.  The result will be checked in some way to determine whether the
                          016A  ;           result is nonzero so that the Z-bit can have its correct setting.
                          016A  ;-
                          016A
                          016A  ADD_PACKED_BYTE_STRING:
                          016A
                          016A          MARK_POINT      ADD_SUB_BSBW_24
            56    71  9A  016A          MOVZBL  -(R1),R6                ; Get byte from first string
                          016D          MARK_POINT      ADD_SUB_BSBW_24
            57    73  9A  016D          MOVZBL  -(R3),R7                ; Get byte from second string
                          0170
                          0170  VAX$ADD_PACKED_BYTE_R6_R7::             ; ASHP also uses this routine
                          0170  ADD_PACKED_BYTE_R6_R7:
56    0000'CF46       90  0170          MOVB    DECIMAL$PACKED_TO_BINARY_TABLE[R6],-
                          0176                  R6                      ; Convert digits to binary
57    0000'CF47       90  0176          MOVB    DECIMAL$PACKED_TO_BINARY_TABLE[R7],-
                          017C                  R7                      ; Convert digits to binary
            57    56  80  017C          ADDB    R6,R7                   ; Form their sum
            57    58  80  017F          ADDB    R8,R7                   ; Add CARRY from last step
                  58  94  0182          CLRB    R8                      ; Assume no CARRY this time
         63 8F    57  91  0184          CMPB    R7,#99                  ; Check for CARRY
                  07  1B  0188          BLEQU   10$                     ; Branch if within bounds
            58    01  90  018A          MOVB    #1,R8                   ; Propogate CARRY to next step
         57    64 8F  82  018D          SUBB    #100,R7                 ; Put R7 into interval 0..99
75    0000'CF47       90  0191  10$:    MOVB    DECIMAL$BINARY_TO_PACKED_TABLE[R7],-
                          0197                  -(R5)                   ; Store converted sum byte
                      05  0197          RSB
```

```
0198          .SUBTITLE       SUBTRACT_PACKED - Subtract Two Packed Decimal Strings
0198  ;+
0198  ; Functional Description:
0198  ;
0198  ;       This routine takes two packed decimal strings whose descriptors
0198  ;       are passed as input parameters, subtracts one string from the
0198  ;       other, and places their sum into another (perhaps identical)
0198  ;       packed decimal string.
0198  ;
0198  ;       At the present time, the result is placed into a 16-byte storage
0198  ;       area while the difference is being evaluated. This drastically reduces
0198  ;       the number of different cases that must be dealt with as each
0198  ;       pair of bytes in the two input strings is added.
0198  ;
0198  ;       The signs of the two input strings have already been dealt with so
0198  ;       this routine performs subtraction in all cases, even if the original
0198  ;       entry was at ADDP4 or ADDP6.
0198  ;
0198  ; Input Parameters:
0198  ;
0198  ;       R0<4:0> - Number of digits in first input decimal string
0198  ;       R1      - Address of least significant digit of first input
0198  ;                 decimal string (the byte containing the  ign)
0198  ;
0198  ;       R2<4:0> - Number of digits in second input decimal string
0198  ;       R3      - Address of least significant digit of second input
0198  ;                 decimal string (the byte containing the sign)
0198  ;
0198  ;       R4<4:0> - Number of digits in output decimal string
0198  ;       R5      - Address of one byte beyond least significant digit of
0198  ;                 intermediate string stored on the stack
0198  ;
0198  ;       R6<3:0> - Sign of first input string in preferred form
0198  ;       R7<3:0> - Sign of second input string in preferred form
0198  ;
0198  ;       R11     - Saved PSL (Z-bit is set, other condition codes are clear)
0198  ;
0198  ;       (SP)    - Saved R5, address of least significant digit of ultimate
0198  ;                 destination string.
0198  ;       4(SP)   - Beginning of 20-byte buffer to hold intermediate result
0198  ;
0198  ; Output Parameters:
0198  ;
0198  ;       The particular input operation (ADDPx or SUBPx) is completed in
0198  ;       this routine. See the routine headers for the four routines that
0198  ;       request addition or subtraction for a list of output parameters
0198  ;       from this routine.
0198  ;
0198  ; Algorithm for Choice of Sign:
0198  ;
0198  ;       The choice of sign for the output string is not nearly so
0198  ;       straightforward as it is in the case of addition. One approach that is
0198  ;       often taken is to make a reasonable guess at the sign of the result.
0198  ;       If the final subtraction causes a BORROW, then the choice was incorrect.
0198  ;       The sign must be changed and the result must be replaced by its tens
0198  ;       complement.
0198  ;
```

```
                        0198  ;       This routine does not guess. Instead, it chooses the input string of
                        0198  ;       the larger absolute magnitude as the minuend for this internal
                        0198  ;       routine and chooses its sign as the sign of the result.
                        0198  ;       This algorithm is actually more efficient than the reasonable
                        0198  ;       guess method and is probably better than a guess method that is never
                        0198  ;       wrong. All complete bytes that are processed in the sign evaluation
                        0198  ;       preprocessing loop are eliminated from consideration in the
                        0198  ;       subtraction loop, which has a higher cost per byte.
                        0198  ;
                        0198  ;       The actual algorithm is as follows. (Note that both input strings have
                        0198  ;       already had leading zeros stripped so their lengths reflect
                        0198  ;       significant digits.)
                        0198  ;
                        0198  ;       1.  If the two strings have unequal lengths, then choose the sign of
                        0198  ;           the string that has the longer length.
                        0198  ;
                        0198  ;       2.  For strings of equal length, choose the sign of the string whose
                        0198  ;           most significant byte is larger in magnitude.
                        0198  ;
                        0198  ;       3.  If the most significant bytes test equal, then decrease the
                        0198  ;           lengths of each string by one byte, drop the previous most
                        0198  ;           significant bytes, and go back to step 2.
                        0198  ;
                        0198  ;       4.  If the two strings test equal, it is not necessary to do any
                        0198  ;           subtraction. The result is identically zero.
                        0198  ;
                        0198  ;       Note that the key to this routine's efficiency is that high order
                        0198  ;       bytes that test equal in this loop are dropped from consideration in
                        0198  ;       the more complicated subtraction loop.
                        0198  ;-
                        0198
                        0198  SUBTRACT_PACKED:
   50   04   01   EF    0198          EXTZV   #1,#4,R0,R0             ; Convert digit count to byte count
             50         019C
   52   04   01   EF    019D          EXTZV   #1,#4,R2,R2             ; Do it for both strings
             52         01A1
        52   50   D1    01A2          CMPL    R0,R2                   ; We want to compare the byte counts
             3C   1F    01A5          BLSSU   40$                     ; R0/R1 represent the smaller string
             2A   1A    01A7          BGTRU   30$                     ; R2/R3 represent the smaller string
                        01A9
                        01A9  ; The two input strings have an equal number of bytes. Compare magnitudes to
                        01A9  ; determine which string is really larger. If the two strings test equal, then
                        01A9  ; skip the entire subtraction loop.
                        01A9
   58   51   50   C3    01A9          SUBL3   R0,R1,R8                ; Point R8 to low address end of R0/R1
   59   53   52   C3    01AD          SUBL3   R2,R3,R9                ; Point R9 to low address end of R2/R3
             50   D5    01B1          TSTL    R0                      ; See if both strings have zero bytes
             0C   13    01B3          BEQL    20$                     ; Still need to check low order digit
                        01B5
                        01B5          MARK_POINT      ADD_SUB_24
        89   88   91    01B5  10$:    CMPB    (R8)+,(R9)+             ; Compare most significant bytes
             29   1F    01B8          BLSSU   40$                     ; R0/R1 represent the smaller string
             17   1A    01BA          BGTRU   30$                     ; R2/R3 represent the smaller string
             52   D7    01BC          DECL    R2                      ; Keep R2 in step with R0
        F4   50   F5    01BE          SOBGTR  R0,10$                  ; ... which gets decremented here
                        01C1
                        01C1  ; At this point, we have reduced both input strings to single bytes that
```

```
                   01C1   ; contain a sign "digit" and may contain a digit in the high order nibble
                   01C1   ; if the original digit counts were nonzero.
                   01C1
                   01C1            MARK_POINT      ADD_SUB_24
58   68   0F   8B  01C1   20$:     BICB3   #^B00001111,(R8),R8      ; Look only at digit, ignoring sign
                   01C5            MARK_POINT      ADD_SUB_24
59   69   0F   8B  01C5            BICB3   #^B00001111,(R9),R9      ; Get the digit from the other string
     59   58   91  01C9            CMPB    R8,R9                    ; Compare these digits
          15   1F  01CC            BLSSU   40$                      ; R0/R1 represent the smaller string
          03   1A  01CE            BGTRU   30$                      ; R2/R3 represent the smaller string
                   01D0
                   01D0   ; The two strings have identical magnitudes. Enter the end processing code
                   01D0   ; with the intermediate result unchanged (that is, zero).
                   01D0
        FF6A   31  01D0            BRW     ADD_SUBTRACT_EXIT        ; Join the common completion code
                   01D3
                   01D3   ; The string described by R0 and R1 has the larger magnitude. Choose its sign.
                   01D3   ; Then swap the two string descriptors so that the main subtraction loops
                   01D3   ; always have R2 and R3 describing the larger string. Note that the use of
                   01D3   ; R6 and R7 as scratch leaves R7<31:8> in an UNPREDICTABLE state.
                   01D3
     59   56   90  01D3   30$:     MOVB    R6,R9                    ; Load preferred sign into R9
     56   50   7D  01D6            MOVQ    R0,R6                    ; Save the longer
     50   52   7D  01D9            MOVQ    R2,R0                    ; Store the shorter on R0 and R1
     52   56   7D  01DC            MOVQ    R6,R2                    ; ... and store the longer in R2 and R3
          57   D4  01DF            CLRL    R7                       ; Insure that R7<31:8> is zero
          03   11  01E1            BRB     50$                      ; Continue along common code path
                   01E3
                   01E3   ; The string described by R2 and R3 has the larger magnitude. Choose its sign.
                   01E3
     59   57   90  01E3   40$:     MOVB    R7,R9                    ; Load preferred sign into R9
                   01E6
     52   50   C2  01E6   50$:     SUBL    R0,R2                    ; Make R2 a difference (R2 GEQU 0)
     03   59   E9  01E9            BLBC    R9,60$                   ; Check if sign is negative
     5B   08   88  01EC            BISB    #PSL$M_N,R11             ; ... so the saved N-bit can be set
                   01EF
                   01EF            MARK_POINT      ADD_SUB_24
56   61   0F   8B  01EF   60$:     BICB3   #^B00001111,(R1),R6      ; Get least significant digit to R6
                   01F3            MARK_POINT      ADD_SUB_24
57   63   0F   8B  01F3            BICB3   #^B00001111,(R3),R7      ; Get least significant digit to R7
          58   D4  01F7            CLRL    R8                       ; Start subtracting with BORROW off
        0032   30  01F9            BSBW    SUB_PACKED_BYTE_R6_R7    ; Subtract the two low order digits
                   01FC
                   01FC   ; R0 contains the number of bytes remaining in the smaller string
                   01FC   ; R2 contains the difference in bytes between the two input strings
                   01FC
          50   D5  01FC            TSTL    R0                       ; Does smaller string have any room?
          06   13  01FE            BEQL    80$                      ; Skip loop if no room at all
                   0200
        0025   30  0200   70$:     BSBW    SUB_PACKED_BYTE_STRING   ; Subtract the next two bytes
     FA   50   F5  0203            SOBGTR  R0,70$                   ; Check for end of loop
                   0206
          52   D5  0206   80$:     TSTL    R2                       ; Does one of the strings have more?
          16   13  0208            BEQL    110$                     ; Skip next loops if all done
                   020A
     0D   58   E9  020A   90$:     BLBC    R8,100$                  ; Life is simple if BORROW clear
                   020D
```

```
        56    D4  020D               CLRL    R6                       ; Otherwise, BORROW must propogate
                  020F               MARK_POINT      ADD_SUB_24
     57 73    9A  020F               MOVZBL  -(R3),R7                 ; So subtract BORROW from single string
       0019   30  0212               BSBW    SUB_PACKED_BYTE_R6_R7    ; Use the special entry point
      F2 52   F5  0215               SOBGTR  R2,90$                   ; Check for this string exhausted
         06   11  0218               BRB     110$                     ; Join common completion code
                  021A
                  021A               MARK_POINT      ADD_SUB_24
     75 73    90  021A      100$:    MOVB    -(R3),-(R5)              ; Simply move src to dst if no BORROW
      FA 52   F5  021D               SOBGTR  R2,100$                  ; ... until we're all done
                  0220
                  0220      110$:
                  0220
                  0220      ;;;      ********** BEGIN TEMP **********
                  0220      ;;;
                  0220      ;;;      THE FOLLOWING HALT INSTRUCTION SHOULD BE REPLACED WITH THE CORRECT
                  0220      ;;;      ABORT CODE.
                  0220      ;;;
                  0220      ;;;      THE HALT IS SIMILAR TO THE
                  0220      ;;;
                  0220      ;;;              MICROCODE CANNOT GET HERE
                  0220      ;;;
                  0220      ;;;      ERRORS THAT OTHER IMPLEMENTATIONS USE.
                  0220      ;;;
         58   D5  0220               tstl    r8                       ; If BORROW is set here, we blew it
         01   13  0222               beql    120$                     ; Branch out if OK
              00  0224               halt                             ; This will cause an OPCDEC exception
                  0225      120$:
                  0225      ;;;
                  0225      ;;;      *********** END TEMP ***********
                  0225
       FF15   31  0225               BRW     ADD_SUBTRACT_EXIT        ; Join common completion code
```

```
0228     .SUBTITLE    SUB_PACKED_BYTE - Subtract Two Bytes Containing Decimal Digi
0228 ;+
0228 ; Functional Description:
0228 ;
0228 ;       This routine takes as input two bytes containing decimal digits and
0228 ;       produces a byte containing their difference. This result is stored in
0228 ;       the output string. Each of the input bytes is converted to a binary
0228 ;       number (with a table-driven conversion), the first number is
0228 ;       subtracted from the second, and the difference is converted back to
0228 ;       two decimal digits stored in a byte.
0228 ;
0228 ;       This routine makes no provisions for bytes that contain illegal
0228 ;       decimal digits. We are using the UNPREDICTABLE statement in the
0228 ;       architectural description of the decimal instructions to its fullest.
0228 ;
0228 ;       The bytes that contain a pair of packed decimal digits can either
0228 ;       exist in packed decimal strings located by R1 and R3 or they can
0228 ;       be stored directly in registers. In the former case, the digits must
0228 ;       be extracted from registers before they can be used in later operations
0228 ;       because the difference will be used as an index register.
0228 ;
0228 ; For entry at SUB_PACKED_BYTE_STRING:
0228 ;
0228 ;       Input Parameters:
0228 ;
0228 ;               R1  - Address one byte beyond byte containing subtrahend
0228 ;               R3  - Address one byte beyond byte containing minuend
0228 ;               R5  - Address one byte beyond location to store difference
0228 ;
0228 ;               R8  - BORROW from previous byte (R8 is either 0 or 1)
0228 ;
0228 ;       Implicit Input:
0228 ;
0228 ;               R6 - Scratch
0228 ;               R7 - Scratch
0228 ;
0228 ;       Output Parameters:
0228 ;
0228 ;               R1 - Decreased by one to point to current byte
0228 ;                    in subtrahend string
0228 ;               R3 - Decreased by one to point to current byte
0228 ;                    in minuend string
0228 ;               R5 - Decreased by one to point to current byte
0228 ;                    in difference string
0228 ;
0228 ;               R8 - Either 0 or 1, reflecting whether this most recent
0228 ;                    subtraction resulted in a BORROW from the next byte.
0228 ;
0228 ; For entry at SUB_PACKED_BYTE_R6_R7:
0228 ;
0228 ;       Input Parameters:
0228 ;
0228 ;               R6<7:0>  - Byte containing decimal digit pair for subtrahend
0228 ;               R6<31:8> - MBZ
0228 ;               R7<7:0>  - Byte containing decimal digit pair for minuend
0228 ;               R7<31:8> - MBZ
0228 ;
```

```
                        0228  ;               R5 - Address one byte beyond location to store difference
                        0228  ;
                        0228  ;               R8 - BORROW from subtraction of previous byte
                        0228  ;                     (R8 is either 0 or 1)
                        0228  ;
                        0228  ;       Output Parameters:
                        0228  ;
                        0228  ;               R5 - Decreased by one to point to current byte
                        0228  ;                     in difference string
                        0228  ;
                        0228  ;               R8 - Either 0 or 1, reflecting whether this most recent
                        0228  ;                     subtraction resulted in a BORROW from the next byte.
                        0228  ;
                        0228  ; Side Effects:
                        0228  ;
                        0228  ;       R6 and R7 are modified by this routine
                        0228  ;
                        0228  ;       R0, R2, R4, and R9 (and, of course, R10 and R11) are preserved
                        0228  ;       by this routine
                        0228  ;
                        0228  ; Assumptions:
                        0228  ;
                        0228  ;       This routine makes two important assumptions.
                        0228  ;
                        0228  ;       1.  If both of the input bytes contain only legal decimal digits, then
                        0228  ;           it is only necessary to add 100 at most once to put all
                        0228  ;           possible differences in the range 0..99. That is,
                        0228  ;
                        0228  ;               0 - 99 - 1 = -100
                        0228  ;
                        0228  ;       2.  The result will be checked in some way to determine whether the
                        0228  ;           result is nonzero so that the Z-bit can have its correct setting.
                        0228  ;-
                        0228
                        0228  SUB_PACKED_BYTE_STRING:
                        0228
                        0228          MARK_POINT      ADD_SUB_BSBW_24
        56    71   9A   0228          MOVZBL  -(R1),R6                ; Get byte from first string
                        022B          MARK_POINT      ADD_SUB_BSBW_24
        57    73   9A   022B          MOVZBL  -(R3),R7                ; Get byte from second string
                        022E
                        022E  SUB_PACKED_BYTE_R6_R7:
56  0000'CF46      90   022E          MOVB    DECIMAL$PACKED_TO_BINARY_TABLE[R6],-
                        0234                  R6                      ; Convert digits to binary
57  0000'CF47      90   0234          MOVB    DECIMAL$PACKED_TO_BINARY_TABLE[R7],-
                        023A                  R7                      ; Convert digits to binary
        57    56   82   023A          SUBB    R6,R7                   ; Form their difference
        57    58   82   023D          SUBB    R8,R7                   ; Include BORROW from last step
              04   19   0240          BLSS    10$                     ; Branch if need to BORROW
              58   94   0242          CLRB    R8                      ; No BORROW next time
              07   11   0244          BRB     20$                     ; Join common exit code
                        0246
     57    64 8F   80   0246  10$:    ADDB    #100,R7                 ; Put R7 into interval 0..99
        58    01   90   024A          MOVB    #1,R8                   ; Propogate BORROW to next step
                        024D
75  0000'CF47      90   024D  20$:    MOVB    DECIMAL$BINARY_TO_PACKED_TABLE[R7],-
                        0253                  -(R5)                   ; Store converted sum byte
```

```
05  0253  1038          RSB
```

```
                    0254          .SUBTITLE       STORE_RESULT - Store Decimal String
                    0254  ;+
                    0254  ; Functional Description:
                    0254  ;
                    0254  ;       This routine takes a packed decimal string that typically contains
                    0254  ;       the result of an arithmetic operation and stores it in another
                    0254  ;       decimal string whose descriptor is specified as an input parameter
                    0254  ;       to the original arithmetic operation.
                    0254  ;
                    0254  ;       The string is stored from the high address end (least significant
                    0254  ;       digits) to the low address end (most significant digits). This order
                    0254  ;       allows all of the special cases to be handled in the simplest fashion.
                    0254  ;
                    0254  ; Input Parameters:
                    0254  ;
                    0254  ;       R1        - Address one byte beyond high address end of input string
                    0254  ;                   (Note that this string must be at least 17 bytes long.)
                    0254  ;
                    0254  ;       R4<4:0> - Number of digits in ultimate destination
                    0254  ;       R5        - Address one byte beyond destination string
                    0254  ;
                    0254  ;       R11       - Contains saved condition codes
                    0254  ;
                    0254  ; Implicit Input:
                    0254  ;
                    0254  ;       The input string must be at least 17 bytes long to contain a potential
                    0254  ;       carry out of the highest digit when doing an add of two large numbers.
                    0254  ;       This carry out of the last byte will be detected and reported as a
                    0254  ;       decimal overflow, either as an exception or simply by setting the V-bit.
                    0254  ;
                    0254  ;       The least significant digit (highest addressed byte) cannot contain a
                    0254  ;       sign digit because that would cause the Z-bit to be incorrectly cleared.
                    0254  ;
                    0254  ; Output Parameters:
                    0254  ;
                    0254  ;       R11<PSL$V_Z> - Cleared if a nonzero digit is stored in output string
                    0254  ;       R11<PSL$V_V> - Set if a nonzero digit is detected after the output
                    0254  ;                      string is exhausted
                    0254  ;
                    0254  ;       A portion of the result (dictated by the size of R4 on input) is
                    0254  ;       moved to the destination string.
                    0254  ;-
                    0254
                    0254  STORE_RESULT:
          54    D6  0254          INCL    R4                      ; Want number of "complete" bytes in
  54   FF 8F    78  0256          ASHL    #-1,R4,R0               ;  output string
          50        025A
          0B    13  025B          BEQL    30$                     ; Skip first loop if none
                    025D
                    025D          MARK_POINT      ADD_SUB_BSBW_24
     75   71    90  025D  10$:    MOVB    -(R1),-(R5)             ; Move the next complete byte
          03    13  0260          BEQL    20$                     ; Check whether to clear Z-bit
     5B   04    8A  0262          BICB    #PSL$M_Z,R11            ; Clear Z-bit if nonzero
     F5   50    F5  0265  20$:    SOBGTR  R0,10$                  ; Keep going?
                    0268
     10   54    E9  0268  30$:    BLBC    R4,50$                  ; Was original R4 odd? Branch if yes
                    026B          MARK_POINT      ADD_SUB_BSBW_24
```

```
  71    F0 8F   8B 026B        BICB3   #^B11110000,-(R1),-(R5) ; If R4 was even, store half a byte
           75      026F
           03   13 0270        BEQL    40$                     ; Need to check for zero here, too
     5B    04   8A 0272        BICB    #PSL$M_Z,R11            ; Clear Z-bit if nonzero
                   0275        MARK_POINT      ADD_SUB_BSBW_24
  61    F0 8F   93 0275 40$:   BITB    #^B11110000,(R1)        ; If high order nibble is nonzero,
           13   12 0279        BNEQ    70$                     ; ... then overflow has occurred
                   027B
                   027B ; The entire destination has been stored. We must now check whether any of
                   027B ; the remaining input string is nonzero and set the V-bit if nonzero is
                   027B ; detected. Note that at least one byte of the output string has been examined
                   027B ; in all cases already. This makes the next byte count calculation correct.
                   027B
           54   D7 027B 50$:   DECL    R4                      ; Restore R4 to its original self
  54    04 01   EF 027D        EXTZV   #1,#4,R4,R0             ; Extract a byte count
           50      0281
  50    10 50   83 0282        SUBB3   R0,#16,R0               ; Loop count is 16 minus byte count
                   0286
                   0286 ; Note that the loop count can never be zero because we are testing a 17-byte
                   0286 ; string and the largest output string can be 16 bytes long.
                   0286
                   0286        MARK_POINT      ADD_SUB_BSBW_24
           71   95 0286 60$:   TSTB    -(R1)                   ; Check next byte for nonzero
           04   12 0288        BNEQ    70$                     ; Nonzero means overflow has occurred
        F9 50   F5 028A        SOBGTR  R0,60$                  ; Check for end of this loop
                   028D
                05 028D        RSB                             ; This is return path for no overflow
                   028E
     5B    02   88 028E 70$:   BISB    #PSL$M_V,R11            ; Indicate that overflow has occurred
                05 0291        RSB                             ; ... and return to the caller
```

```
;LJK0045        0292            .SUBTITLE - CHECK_WRITE_ACCESS - Check Writability of Decimal String
;LJK0045        0292    ;+
;LJK0045        0292    ; Functional Description:
;LJK0045        0292    ;
;LJK0045        0292    ;       The ADDP4 and SUBP4 instructions are unique in that they are the only
;LJK0045        0292    ;       two instructions that read and write the same packed decimal string.
;LJK0045        0292    ;       They are, in fact, implemented as ADDP6 and SUBP6 where the second
;LJK0045        0292    ;       input string, ADDend2 or MINuend, and the result string, SUM or
;LJK0045        0292    ;       DIFference, are the same. But ADDP6 and SUBP6, as well as all other
;LJK0045        0292    ;       packed decimal instructions except ADDP4 and SUBP4, produce
;LJK0045        0292    ;       UNPREDICTABLE results when an output string overlaps any input string.
;LJK0045        0292    ;       With this interpretation, ADDP4 and SUBP4 are the only packed decimal
;LJK0045        0292    ;       instructions that permit overlapping packed decimal strings. (Note
;LJK0045        0292    ;       that the result string, SUM or DIFference, may not overlap the first
;LJK0045        0292    ;       input string, ADDend or SUBtrahend.)
;LJK0045        0292    ;
;LJK0045        0292    ;       The implementation of ADDP4 and SUBP4, interpreted as ADDP6 and SUBP6
;LJK0045        0292    ;       with overlapping strings, needs to protect itself from modifying
;LJK0045        0292    ;       memory until the entire instruction can execute to completion.
;LJK0045        0292    ;       Otherwise, the instruction will be restarted with a different initial
;LJK0045        0292    ;       state than it first had. This is accomplished by probing the result
;LJK0045        0292    ;       string for write access before execution begins. This routine receives
;LJK0045        0292    ;       control if that PROBEW fails.
;LJK0045        0292    ;
;LJK0045        0292    ;       In the interest of simplicity, the PROBEW that is executed always uses
;LJK0045        0292    ;       16 as the byte count of the result string. This routine can then be
;LJK0045        0292    ;       called not only when the output string is inaccessible but also when
;LJK0045        0292    ;
;LJK0045        0292    ;               The output string is shorter than 16 bytes.
;LJK0045        0292    ;
;LJK0045        0292    ;               The address of the output string is within 16 bytes of the
;LJK0045        0292    ;               end of the page.
;LJK0045        0292    ;
;LJK0045        0292    ;               The page containing the string is writable.
;LJK0045        0292    ;
;LJK0045        0292    ;               The next page is not writable.
;LJK0045        0292    ;
;LJK0045        0292    ;       This routine distinguishes inaccessible strings from this rare case of
;LJK0045        0292    ;       a PROBEW failure.
;LJK0045        0292    ;
;LJK0045        0292    ;       In other words, this routine checks the write accessibility of packed
;LJK0045        0292    ;       decimal strings. If the entire string is writable, control is passed
;LJK0045        0292    ;       back to the caller where emulation continues. If the string is not
;LJK0045        0292    ;       writable, an access violation is generated.
;LJK0045        0292    ;
;LJK0045        0292    ; Input Parameters:
;LJK0045        0292    ;
;LJK0045        0292    ;       R4 - Digit count of result string
;LJK0045        0292    ;       R5 - Address of most significant digit in result string
;LJK0045        0292    ;
;LJK0045        0292    ; Output Parameters:
;LJK0045        0292    ;
;LJK0045        0292    ;       None
;LJK0045        0292    ;
;LJK0045        0292    ; Implicit Ouptut:
;LJK0045        0292    ;
;LJK0045        0292    ;       If the string with its correct byte count is writable, control
```

```
;LJK0045                     0292  ;          is returned to the caller.
;LJK0045                     0292  ;
;LJK0045                     0292  ;          If the string with its correct digit count is not writable, then the
;LJK0045                     0292  ;          string is accessed to force an access violation to occur. This will
;LJK0045                     0292  ;          cause control to be transferred eventually to the access violation
;LJK0045                     0292  ;          handler at the end of this module that will back out the instruction
;LJK0045                     0292  ;          so that it can be restarted.
;LJK0045                     0292  ;
;LJK0045                     0292  ; Assumption:
;LJK0045                     0292  ;
;LJK0045                     0292  ;          This routine assumes that the check for illegal digit count has
;LJK0045                     0292  ;          already been made so that R4 is between 0 and 31 inclusive.
;LJK0045                     0292  ;
;LJK0045                     0292  ; Note:
;LJK0045                     0292  ;
;LJK0045                     0292  ;          It is necessary to actually touch the inaccessible page so that the
;LJK0045                     0292  ;          correct reason mask is generated. Logic that uses the PROBEW
;LJK0045                     0292  ;          instruction can determine the inaccessible virtual address but is
;LJK0045                     0292  ;          unable to distinguish, for example, between length violations and page
;LJK0045                     0292  ;          protection violations.
;LJK0045                     0292  ;-
;LJK0045                     0292
;LJK0045                     0292  CHECK_WRITE_ACCESS:
;LJK0045              54 DD  0292           PUSHL   R4                      ; Save the digit count
;LJK0045      54  04  01 EF  0294           EXTZV   #1,#4,R4,R4             ; Convert digits to bytes
                      54     0298
;LJK0045              54 D6  0299           INCL    R4                      ; Count the byte that contains the sign
;LJK0045      65  54  00 0D  029B           PROBEW  #0,R4,(R5)              ; Check writability of the string
;LJK0045              04 13  029F           BEQL    10$                     ; Branch if string cannot be written
;LJK0045          54  8E D0  02A1           MOVL    (SP)+,R4                ; Restore saved R4
;LJK0045                 05  02A4           RSB                             ; Return to caller. String is OK
;LJK0045                     02A5
;LJK0045                     02A5  ; The first and last bytes of the string are touched (written) with an
;LJK0045                     02A5  ;
;LJK0045                     02A5  ;          ADDB2   #0,xxx
;LJK0045                     02A5  ;
;LJK0045                     02A5  ; instruction. This instruction causes the correct access violation reason
;LJK0045                     02A5  ; mask to be generated but does not modify the contents of locations that are
;LJK0045                     02A5  ; accessible. Note that at least one of the following two ADDB2 instructions
;LJK0045                     02A5  ; is guaranteed to generate an access violation, transferring control in a
;LJK0045                     02A5  ; rather complicated way to the ADD_SUB_BSBW_4 access violation handler.
;LJK0045                     02A5
;LJK0045                     02A5           MARK_POINT      ADD_SUB_BSBW_4
;LJK0045          65  00  80 02A5  10$:     ADDB2   #0,(R5)                 ; Touch the first byte
;LJK0045                     02A8           MARK_POINT      ADD_SUB_BSBW_4
;LJK0045        6544  00  80 02A8           ADDB2   #0,(R5)[R4]             ; Touch the last byte
;LJK0045                     02AC
;LJK0045                     02AC  ; We should never reach here unless the PROBE instruction is broken. We
;LJK0045                     02AC  ; will leave this code path in for now but remove it at the same time we
;LJK0045                     02AC  ; change the other two HALT instructions in this module into software
;LJK0045                     02AC  ; generated machine checks exceptions.
;LJK0045                     02AC
;LJK0045                     02AC  ;;; ********** BEGIN TEMP **********
;LJK0045                     02AC  ;;;
;LJK0045                     02AC  ;;; THE FOLLOWING HALT INSTRUCTION SHOULD BE REPLACED WITH THE CORRECT
;LJK0045                     02AC  ;;; ABORT CODE.
;LJK0045                     02AC  ;;;
```

```
;LJK0045          00  02AC  .114            halt                          ; This will cause an OPCDEC exception
;LJK0045              02AD  .115 ;;;
;LJK0045              02AD  .116 ;;; *********** END TEMP ***********
```

```
02AD            .SUBTITLE       VAX$MULP - Multiply Packed
02AD    ;+
02AD    ; Functional Description:
02AD    ;
02AD    ;       The multiplicand string specified by the multiplicand length and
02AD    ;       multiplicand address operands is multiplied by the multiplier string
02AD    ;       specified by the multiplier length and multiplier address operands.  The
02AD    ;       product string specified by the product length and product address
02AD    ;       operands is replaced by the result.
02AD    ;
02AD    ; Input Parameters:
02AD    ;
02AD    ;       R0 - mulrlen.rw         Number of digits in multiplier string
02AD    ;       R1 - mulraddr.ab        Address of multiplier string
02AD    ;       R2 - muldlen.rw         Number of digits in multiplicand string
02AD    ;       R3 - muldaddr.ab        Address of multiplicand string
02AD    ;       R4 - prodlen.rw         Number of digits in product string
02AD    ;       R5 - prodaddr.ab        Address of product string
02AD    ;
02AD    ; Output Parameters:
02AD    ;
02AD    ;       R0 = 0
02AD    ;       R1 = Address of the byte containing the most significant digit of
02AD    ;            the multiplier string
02AD    ;       R2 = 0
02AD    ;       R3 = Address of the byte containing the most significant digit of
02AD    ;            the multiplicand string
02AD    ;       R4 = 0
02AD    ;       R5 = Address of the byte containing the most significant digit of
02AD    ;            the string containing the product
02AD    ;
02AD    ; Condition Codes:
02AD    ;
02AD    ;       N <- product string LSS 0
02AD    ;       Z <- product string EQL 0
02AD    ;       V <- decimal overflow
02AD    ;       C <- 0
02AD    ;
02AD    ; Register Usage:
02AD    ;
02AD    ;       This routine uses all of the general registers. The condition codes
02AD    ;       are computed at the end of the instruction as the final result is
02AD    ;       stored in the product string. R11 is used to record the condition
02AD    ;       codes.
02AD    ;
02AD    ; Notes:
02AD    ;
02AD    ;    1. This routine uses a large amount of stack space to allow storage of
02AD    ;       intermediate results in a convenient form. Specifically, each digit
02AD    ;       pair of the longer input string is stored in binary in a longword on
02AD    ;       the stack. In addition, 32 longwords are set aside to hold the product
02AD    ;       intermediate result. Each longword contains a binary number between 0
02AD    ;       and 99.
02AD    ;
02AD    ;       After the multiplication is complete. Each longword is removed from
02AD    ;       the stack, converted to a packed decimal pair, and stored in the
02AD    ;       output string. Any nonzero cells remaining on the stack after the
```

```
                          02AD  ;       output string has been completely filled are the indication of decimal
                          02AD  ;       overflow.
                          02AD  ;
                          02AD  ;       The purpose of this method of storage is to avoid decimal/binary or
                          02AD  ;       even byte/longword conversions during the calculation of intermediate
                          02AD  ;       results.
                          02AD  ;
                          02AD  ;    2. Trailing zeros are removed from the larger string. All zeros in
                          02AD  ;       the shorter string are eliminated in the sense that no arithmetic
                          02AD  ;       is performed. The output array pointer is simply advanced to point
                          02AD  ;       to the next higher array element.
                          02AD  ;-
                          02AD
                          02AD  VAX$MULP::
          0FFF 8F     BB  02AD          PUSHR   #^M<R0,R1,R2,R3,R4,R5,R6,R7,R8,R9,R10,R11>      ; Save the lot
                          02B1
                          02B1          ESTABLISH_HANDLER       -       ; Store address of access
                          02B1                  ARITH_ACCVIO            ;  violation handler
                          02B6
                          02B6          ROPRAND_CHECK   R4                              ; Insure that R4 is LEQU 31
                          02C1
                          02C1          ROPRAND_CHECK   R2                              ; Insure that R2 is LEQU 31
                          02C9          MARK_POINT      MULP_BSBW_0
             FD34'    30  02C9          BSBW    DECIMAL$STRIP_ZEROS_R2_R3               ; Strip high order zeros from R2/R3
                          02CC
                          02CC          ROPRAND_CHECK   R0                              ; Insure that R0 is LEQU 31
                          02D4          MARK_POINT      MULP_BSBW_0
             FD29'    30  02D4          BSBW    DECIMAL$STRIP_ZEROS_R0_R1               ; Strip high order zeros from R0/R1
                          02D7
    50   04   01      EF  02D7          EXTZV   #1,#4,R0,R0             ; Convert digit count to byte count
              50          02DB
              50      D6  02DC          INCL    R0                      ; Include least significant digit
                          02DE
    52   04   01      EF  02DE          EXTZV   #1,#4,R2,R2             ; Convert digit count to byte count
              52          02E2
              52      D6  02E3          INCL    R2                      ; Include least significant digit
                          02E5
         52   50      D1  02E5          CMPL    R0,R2                   ; See which string is larger
              08      1A  02E8          BGTRU   3$                      ; R2/R3 describes the longer string
         58   52      7D  02EA          MOVQ    R2,R8                   ; R8 and R9 describe the longer string
         7E   50      7D  02ED          MOVQ    R0,-(SP)                ; Shorter string descriptor also saved
              06      11  02F0          BRB     6$
                          02F2
         58   50      7D  02F2  3$:     MOVQ    R0,R8                   ; R8 and R9 describe the longer string
         7E   52      7D  02F5          MOVQ    R2,-(SP)                , Shorter string descriptor also saved
                          02F8
                          02F8  ; Create space for the output array on the stack (32 longwords of zeros)
                          02F8
         50   08      D0  02F8  6$:     MOVL    #8,R0                   ; Eight pairs of quadwords
                          02FB
              7E      7C  02FB  10$:    CLRQ    -(SP)                   ; Clear one pair
              7E      7C  02FD          CLRQ    -(SP)                   ; ... and another
           F9 50      F5  02FF          SOBGTR  R0,10$                  ; Do all eight pairs
                          0302
         57   5E      D0  0302          MOVL    SP,R7                   ; Store beginning of output array in R7
                          0305
                          0305  ; The longer input array will be stored on the stack as an array of
```

I 9

```
                        0305  ; longwords. Each array element contains a number between 0 and 99,
                        0305  ; representing a pair of digits in the original packed decimal string.
                        0305  ; Because the units digit is stored with the sign in packed decimal format,
                        0305  ; it is necessary to shift the number as we store it. This is accomplished by
                        0305  ; multiplying the number by ten.
                        0305  ;
                        0305  ; The longer array is described by R8 (byte count) and R9 (address of most
                        0305  ; significant digit pair).
                        0305
      55    58    59  C1 0305          ADDL3   R9,R8,R5                        ; Point R5 beyond sign digit
            54    58  D0 0309          MOVL    R8,R4                           ; R4 contains the loop count
                        030C
                        030C  ; An array of longwords is allocated on the stack. R3 starts out pointing
                        030C  ; at the longword beyond the top of the stack. The first remainder, guaranteed
                        030C  ; to be zero, is "stored" here. The rest of the digit pairs are stored safely
                        030C  ; below the top of the stack.
                        030C
            53    58  CE 030C          MNEGL   R8,R3                           ; Stack grows toward lower addresses
           5E   6E43  DE 030F          MOVAL   (SP)[R3],SP                     ; Allocate the space
      53    5E    04  C3 0313          SUBL3   #4,SP,R3                        ; Point R3 at next lower longword
                        0317
                        0317           MARK_POINT      MULP_R8
            51    75  9A 0317  20$:    MOVZBL  -(R5),R1                        ; Get next digit pair
      51   0000'CF41  9A 031A          MOVZBL  DECIMAL$PACKED_TO_BINARY_TABLE[R1],-
                        0320                   R1                              ; Convert digits to binary
      52    51    0A  7A 0320          EMUL    #10,R1,R2,R0                    ; Multiply by 10
                  50     0324
       00000064 8F    7B 0325          EDIV    #100,R0,R2,(R3)+                ; Divide by 100
      83    52    50     032B
            E6    54  F5 032E          SOBGTR  R4,20$
                        0331
            63    52  D0 0331          MOVL    R2,(R3)                         ; Store final quotient
            59    5E  D0 0334          MOVL    SP,R9                           ; Remember array address in R9
                6E48  DF 0337          PUSHAL  (SP)[R8]                        ; Store start of fixed size area
                        033A
                        033A  ; Check for trailing zeros in the input array stored on the stack. If any are
                        033A  ; present, they are removed and the product array is adjusted accordingly.
                        033A
                  89  D5 033A  30$:    TSTL    (R9)+                           ; Is next number zero?
                  08  12 033C          BNEQ    40$                             ; Leave loop if nonzero
            57    04  C0 033E          ADDL    #4,R7                           ; Advance output pointer to next element
            F6    58  F5 0341          SOBGTR  R8,30$                          ; Keep going
                        0344
                        0344  ; If we drop through the loop, then the entire input array is zero. There is
                        0344  ; no need to perform any arithmetic because the product will be zero (and the
                        0344  ; output array on the stack starts out as zero). The only remaining work is
                        0344  ; to store the result in the output string and set the condition codes.
                        0344
                  2C  11 0344          BRB     70$                             ; Exit to end processing
                        0346
                        0346  ; Now multiply the input array by each successive digit pair. In order to
                        0346  ; allow R10 to continue to locate ARITH_ACCVIO while we execute this loop, it
                        0346  ; is necessary to perform a small amount of register juggling. In essence,
                        0346  ; R8 and R9 switch the identity of the string that they describe.
                        0346
            59    04  C2 0346  40$:    SUBL    #4,R9                           ; Readjust input array pointer
            7E    58  7D 0349          MOVQ    R8,-(SP)                        ; Save R8/R9 descriptor on stack
```

```
      58    08 AE    D0  034C           MOVL    8(SP),R8                ; Point R8 at start of 32-longword array
   58     0080 C8    7D  0350           MOVQ    <32*4>(R8),R8           ; Get descriptor that follows that array
      59    58       C0  0355           ADDL2   R8,R9                   ; Point R9 beyond sign byte
                         0358
      53    87       DE  0358  50$:     MOVAL   (R7)+,R3                ; Output array address to R3
                         035B           MARK_POINT      MULP_AT_SP
      51    79       9A  035B           MOVZBL  -(R9),R1                ; Next digit pair to R1
 56     0000'CF41    9A  035E           MOVZBL  DECIMAL$PACKED_TO_BINARY_TABLE[R1],-
                         0364                   R6                      ; Convert digits to binary
            06       13  0364           BEQL    60$                     ; Skip the work if zero
      54    6E       7D  0366           MOVQ    (SP),R4                 ; Input array descriptor to R4/R5
          0104       30  0369           BSBW    EXTEND_STRING_MULTIPLY  ; Do the work
         E9 58       F5  036C  60$:     SOBGTR  R8,50$                  ; Any more multiplier digits?
                         036F
      5E    08       C0  036F           ADDL    #8,SP                   ; Discard saved long string descriptor
                         0372
      5E    6E       D0  0372  70$:     MOVL    (SP),SP                 ; Remove input array from stack
                         0375
                         0375  ; At this point, the product string is located in a 32-longword array on
                         0375  ; the top of the stack. Each longword corresponds to a pair of digits in
                         0375  ; the output string. As digits are removed from the stack, they are checked
                         0375  ; for nonzero to obtain the correct setting of the Z-bit. After the output
                         0375  ; string has been filled, the remainder of the product string is removed from
                         0375  ; the stack. If a nonzero result is detected at this stage, the V-bit is set.
                         0375
      59    20       D0  0375           MOVL    #32,R9                  ; Set up array counter
 54       0098 CE    7D  0378           MOVQ    < <32*4> + -            ; Skip over 32-longword array
                         037D                     <2*4>  + -            ;  and saved string descriptor
                         037D                     <4*4> >(SP),R4        ;  to retrieve original R4 and R5
```

```
                        037D          .SUBTITLE       Common Exit Path for VAX$MULP and VAX$DIVP
                        037D  ;+
                        037D  ; The code for VAX$MULP and VAX$DIVP merges at this point. The result is stored
                        037D  ; in an array of longwords at the top of the stack. The size of this array is
                        037D  ; stored in R9. The original R4 and R5 heve been retrieved from the stack.
                        037D  ;
                        037D  ; Input Parameters:
                        037D  ;
                        037D  ;       R4 - Contains byte count of destination string in R4 <1:4>
                        037D  ;       R5 - Address of most significant digit of destination string
                        037D  ;       R9 - Count of longwords in result array on stack
                        037D  ;
                        037D  ;       Contents of result array
                        037D  ;
                        037D  ; Implicit Input:
                        037D  ;
                        037D  ;       Signs of two input factors (multiplier and multiplicand or
                        037D  ;               divisor and dividend)
                        037D  ;-
                        037D
                        037D  MULTIPLY_DIVIDE_EXIT:
              5B    DC  037D          MOVPSL  R11                     ; Get current PSL
  04    00    04    F0  037F          INSV    #PSL$M_Z,#0,#4,R11      ; Clear all codes except Z-bit
              5B        0383
                        0384          ESTABLISH_HANDLER       -       ; Store address of access
                        0384                  ARITH_ACCVIO            ;  violation handler again
  54    04    01    EF  0389          EXTZV   #1,#4,R4,R3             ; Excess byte count to R3
              53        038D
              3B    13  038E          BEQL    125$                    ; Skip to single digit code
  57    55    53    C1  0390          ADDL3   R3,R5,R7                ; Remember address of sign byte
  55    57    01    C1  0394          ADDL3   #1,R7,R5                ; Point R5 beyond end of product string
                        0398
        51    8E    D0  0398  80$:    MOVL    (SP)+,R1                ; Remove next value from stack
              03    13  039B          BEQL    90$                     ; Do not clear Z-bit if zero
        5B    04    8A  039D          BICB2   #PSL$M_Z,R11            ; Clear Z-bit
                        03A0
                        03A0          MARK_POINT      MULP_DIVP_R9
  75    0000'CF41   90  03A0  90$:    MOVB    DECIMAL$BINARY_TO_PACKED_TABLE[R1],-
                        03A6                  -(R5)                   ; Store converted sum byte
              59    D7  03A6          DECL    R9                      ; One less element on the stack
              1C    15  03A8          BLEQ    116$                    ; Exit loop if result array exhausted
        EB    53    F5  03AA          SOBGTR  R3,80$                  ; Keep going?
                        03AD
        22    54    E9  03AD  100$:   BLBC    R4,120$                 ; Different for even digit count
                        03B0
                        03B0  ; The output string consists of an odd number of digits. A complete digit
                        03B0  ; pair can be stored in the most significant (lowest addressed) byte of
                        03B0  ; the product string.
                        03B0
        51    8E    D0  03B0          MOVL    (SP)+,R1                ; Remove next value from stack
              03    13  03B3          BEQL    110$                    ; Do not clear Z-bit if zero
        5B    04    8A  03B5          BICB2   #PSL$M_Z,R11            ; Clear Z-bit
                        03B8
                        03B8          MARK_POINT      MULP_DIVP_R9
  75    0000'CF41   90  03B8  110$:   MOVB    DECIMAL$BINARY_TO_PACKED_TABLE[R1],-
                        03BE                  -(R5)                   ; Store converted sum byte
              59    D7  03BE          DECL    R9                      ; One less element on the stack
```

```
              04    15  03C0           BLEQ    116$                          ; Exit loop if result array exhausted
              38    11  03C2           BRB     140$                          ; Perform overflow check
                        03C4
                        03C4  ; This loop executes if the result array has fewer elements than the output
                        03C4  ; string. The remaining bytes in the output string are filled with zeros.
                        03C4  ; There is no need for an overflow check.
                        03C4
                        03C4           MARK_POINT      MULP_DIVP_8
              75    94  03C4  114$:    CLRB    -(R5)                         ; Store another zero byte
           FB 53    F4  03C6  116$:    SOBGEQ  R3,114$                       ; Any more room in output string
                        03C9
              38    11  03C9           BRB     150$                          ; Determine sign of result
                        03CB
                        03CB  ; This code path is used in the case where the output digit count is 0 or 1.
                        03CB  ; R5 must be advanced
                        03CB
        57    55    D0  03CB  125$:    MOVL    R5,R7                         ; Remember address of output sign byte
              55    D6  03CE           INCL    R5                            ; Advance R5 so common code can be used
              DB    11  03D0           BRB     100$                          ; Join common code path
                        03D2
                        03D2  ; The output string consists of an even number of digits. Only the low order
                        03D2  ; nibble is stored in the most significant (lowest addresses) byte. A zero is
                        03D2  ; stored in the high order nibble. If the high order digit would have been
                        03D2  ; nonzero, the V-bit is set and the overflow check is bypassed because there
                        03D2  ; are faster ways to clean the stack if we do not have to check for nonzero
                        03D2  ; at the same time.
                        03D2
        51    8E    D0  03D2  120$:    MOVL    (SP)+,R1                      ; Remove next value from stack
 51   0000'CF41     90  03D5           MOVB    DECIMAL$BINARY_TO_PACKED_TABLE[R1],-
                        03DB                   R1                            ; Obtain converted sum byte
                        03DB           MARK_POINT      MULP_DIVP_R9
   51    F0 8F      8B  03DB           BICB3   #^XF0,R1,-(R5)                ; Store byte, clearing high order nibble
              75        03DF
              03    13  03E0           BEQL    130$                          ; Do not clear Z-bit if zero
        5B    04    8A  03E2           BICB2   #PSL$M_Z,R11                  ; Clear Z-bit
   51    F0 8F      93  03E5  130$:    BITB    #^XF0,R1                      ; Is high order nibble nonzero?
              06    12  03E9           BNEQ    133$                          ; Yes, go set overflow bit
              59    D7  03EB           DECL    R9                            ; One less element on the stack
              D7    15  03ED           BLEQ    116$                          ; Exit loop if result array exhausted
              0B    11  03EF           BRB     140$                          ; Check rest of result array for nonzero
                        03F1
                        03F1  ; If we detect overflow, we need to adjust R9 to reflect the nonzero longword
                        03F1  ; removed from the stack before we enter the next code block that sets the
                        03F1  ; V-bit and cleans off the stack based on the contents of R9.
                        03F1
              59    D7  03F1  133$:    DECL    R9                            ; One more longword removed from stack
                        03F3
                        03F3  ; A nonzero digit has been discovered in a position that cannot be stored in
                        03F3  ; the output string. Set the V-bit, remove the rest of the product array from
                        03F3  ; the stack, and join the exit processing in the code that determines the sign
                        03F3  ; of the product.
                        03F3
        5B    02    88  03F3  135$:    BISB    #PSL$M_V,R11                  ; Set the overflow bit
      5E    6E49    DE  03F6           MOVAL   (SP)[R9],SP                   ; Clean off remaining product string
              07    11  03FA           BRB     150$                          ; Go to code that determines the sign
                        03FC
                        03FC  ; The remainder of the product array must be removed from the stack. A nonzero
```

```
                    03FC  ; result causes the V-bit to be set and the rest of the loop to be skipped.
                    03FC  ; Note that there is always a nonzero loop count remaining at this point.
                    03FC
          8E    D5  03FC  140$:   TSTL    (SP)+                           ; Is next longword zero?
          F1    12  03FE          BNEQ    133$                            ; No, leave loop
       F9 59    F5  0400          SOBGTR  R9,140$
                    0403
                    0403  ; The final product string has been stored and the V- and Z-bits have their
                    0403  ; correct settings. The sign of the product must be determined from the
                    0403  ; signs of the two input strings. Opposite signs produce a negative product.
                    0403  ; Same signs (in any representation) produce a plus sign in the output string.
                    0403
     5E   08    C0  0403  150$:   ADDL    #8,SP                           ; Discard saved string descriptor
     56   0C    D0  0406          MOVL    #12,R6                          ; Assume final result is positive
     50   6E    7D  0409          MOVQ    (SP),R0                         ; Retrieve original R0/R1 pair
50   04   01    EF  040C          EXTZV   #1,#4,R0,R0                     ; Get byte count for first input string
          50        0410
     51   50    C0  0411          ADDL    R0,R1                           ; Point R1 to byte containing sign
                    0414          MARK_POINT      MULP_DIVP_0
61     F0 8F    8B  0414          BICB3   #^B11110000,(R1),R0             ; R0 contains the sign "digit"
          50        0418
                    0419
                    0419          CASE    R0,TYPE=B,LIMIT=#10,<-          ; Dispatch on sign digit
                    0419                  220$,-                          ; 10 => sign is "+"
                    0419                  210$,-                          ; 11 => sign is "-"
                    0419                  220$,-                          ; 12 => sign is "+"
                    0419                  210$,-                          ; 13 => sign is "-"
                    0419                  220$,-                          ; 14 => sign is "+"
                    0419                  220$,-                          ; 15 => sign is "+"
                    0419                  >
                    0429
     54   01    D0  0429  210$:   MOVL    #1,R4                           ; Count a minus sign
          02    11  042C          BRB     230$                            ; Now check second input sign
                    042E
          54    D4  042E  220$:   CLRL    R4                              ; No real minus signs so far
                    0430
  52   08 AE    7D  0430  230$:   MOVQ    8(SP),R2                        ; Retrieve original R2/R3 pair
52   04   01    EF  0434          EXTZV   #1,#4,R2,R2                     ; Get byte count for second input string
          52        0438
     53   52    C0  0439          ADDL    R2,R3                           ; Point R3 to byte containing sign
                    043C          MARK_POINT      MULP_DIVP_0
63     F0 8F    8B  043C          BICB3   #^B11110000,(R3),R2             ; R2 contains the sign "digit"
          52        0440
                    0441
                    0441          CASE    R2,TYPE=B,LIMIT=#10,<-          ; Dispatch on sign digit
                    0441                  250$,-                          ; 10 => sign is "+"
                    0441                  240$,-                          ; 11 => sign is "-"
                    0441                  250$,-                          ; 12 => sign is "+"
                    0441                  240$,-                          ; 13 => sign is "-"
                    0441                  250$,-                          ; 14 => sign is "+"
                    0441                  250$,-                          ; 15 => sign is "+"
                    0441                  >
                    0451
          54    D6  0451  240$:   INCL    R4                              ; Remember that sign was minus
       09 54    E9  0453  250$:   BLBC    R4,260$                         ; Even parity indicates positive result
    10 5B 02    E0  0456          BBS     #PSL$V_Z,R11,270$               ; Step out of line for minus zero check
       5B 08    88  045A          BISB    #PSL$M_N,R11                    ; Set N-bit in saved PSW
```

```
                    56    D6  045D  1486 255$:   INCL    R6                           ; Change sign to minus
                              045F  1487
                              045F  1488         MARK_POINT       MULP_DIVP_O
        04    00    56    F0  045F  1489 260$:   INSV    R6,#0,#4,(R7)                ; Store sign in result string
                    67        0463
                 10 AE    D4  0464  1490         CLRL    16(SP)                       ; Set saved R4 to zero
                 FB96'    31  0467  1491         BRW     VAX$DECIMAL_EXIT             ; Join common exit code
                              046A  1492
                              046A  1493 ; If the result is negative zero, then it must be changed to positive zero
                              046A  1494 ; unless overflow has occurred, in which case, the sign is left as negative
                              046A  1495 ; but the N-bit is clear.
                              046A  1496
           EF 5B    01    E0  046A  1497 270$:   BBS     #PSL$V_V,R11,255$            ; Make sign negative if overflow
                    EF    11  046E  1498         BRB     260$                         ; Sign will be positive
```

```
0470          .SUBTITLE       EXTEND_STRING_MULTIPLY - Multiply a String by a Number
0470    ;+
0470    ; Functional Description:
0470    ;
0470    ;       This routine multiplies an array of numbers (each array element LEQU
0470    ;       99) by a number (also LEQU 99). The resulting product array is added
0470    ;       to another array, each of whose elements is also LEQU 99.
0470    ;
0470    ; Input Parameters:
0470    ;
0470    ;       R3 - Pointer to output array
0470    ;       R4 - Input array size
0470    ;       R5 - Input array address
0470    ;       R6 - Multiplier
0470    ;
0470    ; Output Parameters:
0470    ;
0470    ;       None
0470    ;
0470    ; Implicit Output:
0470    ;
0470    ;       The output array is altered.
0470    ;
0470    ;       An intermediate product array is produced by multiplying eacn input
0470    ;       array element by the multiplier. Each product array element is then
0470    ;       added to the corresponding output array element.
0470    ;
0470    ; Side Effects:
0470    ;
0470    ;       R3, R4, and R5 are modified by this routine.
0470    ;
0470    ;       R6 is preserved.
0470    ;
0470    ;       R0, R1, and R2 are used as scratch registers. R0 and R1 contain the
0470    ;       quadword result of EMUL that is then passed into EDIV.
0470    ;
0470    ; Assumptions:
0470    ;
0470    ;       This routine assumes that all array elements lie in the range from 0
0470    ;       to 99 inclusive. (This is true if all input strings contain only legal
0470    ;       decimal digits.) The arithmetic performed by this routine will
0470    ;       maintain this assumption. That is,
0470    ;
0470    ;                           input array element    LEQU 99
0470    ;               times                multiplier    LEQU 99
0470    ;                      ------------------------
0470    ;                                       product                 LEQU 99*99
0470    ;               plus                      carry    LEQU 99
0470    ;                      ------------------------
0470    ;                              modified product                 LEQU 99*100
0470    ;               plus   old output array element    LEQU 99
0470    ;                      ------------------------
0470    ;                      new output array element                 LEQU 99*101 = 9999
0470    ;
0470    ;       A number LEQU 9999, when divided by 100, is guaranteed to produce both
0470    ;       a quotient and a remainder LEQU 99.
0470    ;-
```

```
                        0470
                        0470  EXTEND_STRING_MULTIPLY:
              52   D4   0470          CLRL    R2                    ; Initial carry is zero
                        0472
   52   85    56   7A   0472  10$:    EMUL    R6,(R5)+,R2,R0        ; Form modified product (R0 LEQU 9900)
              50        0476
        50    63   C0   0477          ADDL2   (R3),R0               ; Add old output array element
   00000064   8F   7B   047A          EDIV    #100,R0,R2,(R3)+      ; Remainder to output array
   83   52    50        0480
                        0483                                        ; Quotient becomes carry
        EC 54      F5   0483          SOBGTR  R4,10$                ; Keep going?
                        0486
                        0486  ; This remaining code looks more complicated than it actually is. In the
                        0486  ; usual case, the routine exits immediately. In the event that a carry
                        0486  ; occurs, one additional entry in the output array will be modified. Only in
                        0486  ; the rare case of an output array consisting of a string of 99s will any
                        0486  ; significant looping occur.
                        0486
        63    52   C0   0486          ADDL2   R2,(R3)               ; Add final carry
              63   D1   0489  20$:    CMPL    (R3),#100             ; Do we overflow into next digit pair?
   00000064   8F        048B
              01   1E   0490          BGEQU   30$                   ; Branch if carry
                   05   0492          RSB                           ; Otherwise, all done
                        0493
   00000064   8F   C2   0493  30$:    SUBL    #100,(R3)+            ; Readjust entry and advance pointer
              83        0499
              63   D6   049A          INCL    (R3)                  ; Propogate carry
              E8   11   049C          BRB     20$                   ; ... and test this entry for overflow
```

```
049E        .SUBTITLE       VAX$DIVP - Divide Packed
049E  ;+
049E  ; Functional Description:
049E  ;
049E  ;       The dividend string  specified  by  the  dividend  length  and  dividend
049E  ;       address  operands  is  divided  by  the  divisor string specified by the
049E  ;       divisor length  and  divisor  address  operands.   The  quotient  string
049E  ;       specified  by  the  quotient  length  and  quotient  address operands is
049E  ;       replaced by the result.
049E  ;
049E  ; Input Parameters:
049E  ;
049E  ;       R0 - divrlen.rw             Number of digits in divisor string
049E  ;       R1 - divraddr.ab            Address of divisor string
049E  ;       R2 - divdlen.rw             Number of digits in dividend string
049E  ;       R3 - divdaddr.ab            Address of dividend string
049E  ;       R4 - quolen.rw              Number of digits in quotient string
049E  ;       R5 - quoaddr.ab             Address of quotient string
049E  ;
049E  ; Output Parameters:
049E  ;
049E  ;       R0 = 0
049E  ;       R1 = Address of the byte containing the most significant digit of
049E  ;            the divisor string
049E  ;       R2 = 0
049E  ;       R3 = Address of the byte containing the most significant digit of
049E  ;            the dividend string
049E  ;       R4 = 0
049E  ;       R5 = Address of the byte containing the most significant digit of
049E  ;            the string containing the quotient
049E  ;
049E  ; Condition Codes:
049E  ;
049E  ;       N <- quotient string LSS 0
049E  ;       Z <- quotient string EQL 0
049E  ;       V <- decimal overflow
049E  ;       C <- 0
049E  ;
049E  ; Register Usage:
049E  ;
049E  ;       This routine uses all of the general registers. The condition codes
049E  ;       are computed at the end of the instruction as the final result is
049E  ;       stored in the quotient string. R11 is used to record the condition
049E  ;       codes.
049E  ;
049E  ; Algorithm:
049E  ;
049E  ;       This algorithm is the straightforward approach described in
049E  ;
049E  ;               The Art of Computer Programming
049E  ;               Second Edition
049E  ;
049E  ;               Volume 2 / Seminumerical Algorithms
049E  ;               Donald E. Knuth
049E  ;
049E  ;               1981
049E  ;               Addison-Wesley Publishing Company
```

```
                    049E  ;                       Reading, Massachusetts
                    049E  ;
                    049E  ; Notes:
                    049E  ;
                    049E  ;         The choice of a longword array to store the auotient deserves a
                    049E  ;         comment. In VAX$MULP, a longword array was used because its elements
                    049E  ;         were used directly by MULP and DIVP instructions. The use of longwords
                    049E  ;         eliminated the need to convert back and forth between longwords and
                    049E  ;         bytes. In this routine, the QUOTIENT_DIGIT routine returns its result
                    049E  ;         in a register, which result can easily be stored in whatever way is
                    049E  ;         convenient. By using longwords instead of bytes, this routine can use
                    049E  ;         the same end processing code as MULP, a sizeable savings in code.
                    049E  ;-
                    049E
                    049E          .ENABLE         LOCAL_BLOCK
                    049E
                    049E  ;+
                    049E  ; This code path is entered if the divisor is zero.
                    049E  ;
                    049E  ; Input Parameter:
                    049E  ;
                    049E  ;         (SP) - Return PC
                    049E  ;
                    049E  ; Output Parameters:
                    049E  ;
                    049E  ;         0(SP) - SRM$K_FLT_DIV_T (Arithmetic trap code)
                    049E  ;         4(SP) - Final state PSL
                    049E  ;         8(SP) - Return PC
                    049E  ;
                    049E  ; Implicit Output:
                    049E  ;
                    049E  ;         Control passes through this code to VAX$REFLECT_TRAP.
                    049E  ;-
                    049E
                    049E  DIVIDE_BY_ZERO:
 0FFF 8F     BA     049E          POPR    #^M<R0,R1,R2,R3,R4,R5,R6,R7,R8,R9,R10,R11>
                    04A2                                          ; Restore registers and reset SP
      7E     DC     04A2          MOVPSL  -(SP)                   ; Save final PSL on stack
      04     DD     04A4          PUSHL   #SRM$K_FLT_DIV_T        ; Store arithmetic trap code
   FB57'     31     04A6          BRW     VAX$REFLECT_TRAP        ; Report exception
                    04A9
                    04A9  ; If the divisor contains more nonzero digits than the dividend, then the
                    04A9  ; quotient will be identically zero. Set up the stack and the registers (R4,
                    04A9  ; R5, and R9)  so that the exit code will be entered to produce this result.
                    04A9
      7E     D4     04A9  1$:     CLRL    -(SP)                   ; Fake a quotient digit
 59   01     D0     04AB          MOVL    #1,R9                   ; Count that digit
    FECC     31     04AE          BRW     MULTIPLY_DIVIDE_EXIT    ; Store the zero in the output string
                    04B1
                    04B1  VAX$DIVP::
 0FFF 8F     BB     04B1          PUSHR   #^M<R0,R1,R2,R3,R4,R5,R6,R7,R8,R9,R10,R11>     ; Save the lot
                    04B5
                    04B5          ESTABLISH_HANDLER       -       ; Store address of access
                    04B5                  ARITH_ACCVIO            ;  violation handler
                    04BA
                    04BA          ROPRAND_CHECK   R4                      ; Insure that R4 is LEQU 31
                    04C5
```

```
                              04C5          ROPRAND_CHECK  R2                       ; Insure that R2 is LEQU 31
                              04CD          MARK_POINT     DIVP_BSBW_0
              FB30'    30     04CD          BSBW    DECIMAL$STRIP_ZEROS_R2_R3       ; Strip high order zeros from R2/R3
                              04D0
                              04D0          ROPRAND_CHECK  R0                       ; Insure that R0 is LEQU 31
                              04D8          MARK_POINT     DIVP_BSBW_0
              FB25'    30     04D8          BSBW    DECIMAL$STRIP_ZEROS_R0_R1       ; Strip high order zeros from R0/R1
                              04DB
                              04DB  ; Insure that the divisor is not zero. Because leading zeros have already
                              04DB  ; been eliminated, the divisor can only be zero if R0 is 0 (zero length
                              04DB  ; strings are identically zero) or 1 (R1 contains a sign digit in the low
                              04DB  ; order nibble and zero in the high order nibble). Note that an exception
                              04DB  ; will not be generated if an even length string has an illegal nonzero digit
                              04DB  ; stored in its most significant nibble (including an illegal form of a zero
                              04DB  ; length string.
                              04DB
  50     04     01     EF     04DB          EXTZV   #1,#4,R0,R0                     ; Convert divisor digit count to bytes
                50            04DF
                06     12     04E0          BNEQ    10$                             ; Skip zero divisor check unless zero
                              04E2          MARK_POINT     DIVP_0
     61     F0  8F     93     04E2          BITB    #^B11110000,(R1)                ; Check for zero in ones digit
                B6     13     04E6          BEQL    DIVIDE_BY_ZERO                  ; Generate exception if zero
                              04E8
                              04E8  ; This routine chooses to do its work with a fair amount of internal storage,
                              04E8  ; all of it allocated on the stack. The quotient is stored as it is computed,
                              04E8  ; in a 16-longword array. The dividend and divisor are stored as longword arrays,
                              04E8  ; with each array element storing a digit pair from the original packed
                              04E8  ; decimal string. The numerator digits are shifted by one digit (multiplied
                              04E8  ; by ten) so that the quotient has its digits correctly placed, leaving room
                              04E8  ; for a sign in the low order nibble of the least significant byte. A scratch
                              04E8  ; array is also allocated on the stack to accommodate intermediate results
                              04E8  ; of the QUOTIENT_DIGIT routine.
                              04E8
                50     D6     04E8  10$:    INCL    R0                              ; Include least significant digit
         58     50     7D     04EA          MOVQ    R0,R8                           ; Let R8 and R9 describe the divisor
                              04ED
  52     04     01     EF     04ED          EXTZV   #1,#4,R2,R2                     ; Convert dividend digit count to bytes
                52            04F1
                52     D6     04F2          INCL    R2                              ; Include least significant digit
         7E     52     7D     04F4          MOVQ    R2,-(SP)                        ; Save dividend descriptor on stack
                              04F7
  56     52     50     C3     04F7          SUBL3   R0,R2,R6                        ; Calculate main loop count
                AC     1F     04FB          BLSSU   1$                              ; Quotient will be zero
                56     D6     04FD          INCL    R6                              ; One extra digit is always there
                              04FF
                              04FF  ; Allocate R6 longwords of zero on the stack
                              04FF
         50     56     D0     04FF          MOVL    R6,R0                           ; Let R0 be the loop counter
                7E     D4     0502  15$:    CLRL    -(SP)                           ; Set aside another quotient digit
             FB 50     F5     0504          SOBGTR  R0,15$                          ; Keep going
                              0507
         57     5E     D0     0507          MOVL    SP,R7                           ; Remember where this array starts
                              050A
                              050A  ; The divisor will be stored on the stack as an array of
                              050A  ; longwords. Each array element contains a number between 0 and 99,
                              050A  ; representing a pair of digits in the original packed decimal string.
                              050A  ; Because the units digit is stored with the sign in packed decimal format,
```

```
                         050A  ; it is necessary to shift the number as we store it. This is accomplished by
                         050A  ; multiplying the number by ten.
                         050A  ;
                         050A  ; The divisor string is described by R8 (byte count) and R9 (address of most
                         050A  ; significant digit pair).
                         050A
      55   58   59   C1  050A          ADDL3   R9,R8,R5                  ; Point R5 beyond sign digit
           54   58   D0  050E          MOVL    R8,R4                     ; R4 contains the loop count
                         0511
                         0511  ; Put in an extra digit place for the divisor. This allows several common
                         0511  ; subroutines to be used when operating on the divisor string.
                         0511
                7E   D4  0511          CLRL    -(SP)                     ; Set aside a place holder
                         0513
                         0513  ; An array of longwords is allocated on the stack. R3 starts out pointing
                         0513  ; at the longword beyond the top of the stack. The first remainder, guaranteed
                         0513  ; to be zero, is "stored" here. The rest of the digit pairs are stored safely
                         0513  ; below the top of the stack.
                         0513
           53   58   CE  0513          MNEGL   R8,R3                     ; Stack grows toward lower addresses
        5E    6E43   DE  0516          MOVAL   (SP)[R3],SP               ; Allocate the space
      53   5E   04   C3  051A          SUBL3   #4,SP,R3                  ; Point R3 at next lower longword
                         051E
                         051E          MARK_POINT      DIVP_R6_R7
           51   75   9A  051E  20$:    MOVZBL  -(R5),R1                  ; Get next digit pair
      51    0000'CF41    9A  0521          MOVZBL  DECIMAL$PACKED_TO_BINARY_TABLE[R1],-
                         0527                  R1                        ; Convert digits to binary
      52   51   0A   7A  0527          EMUL    #10,R1,R2,R0              ; Multiply by 10
                50       052B
        00000064 8F  7B  052C          EDIV    #100,R0,R2,(R3)+          ; Divide by 100
      83   52   50       0532
           E6   54   F5  0535          SOBGTR  R4,20$
                         0538
                         0538  ; There are two cases where the final quotient (contents of R2) is zero.
                         0538  ; In these cases, the number of nonzero digit pairs in the divisor array is
                         0538  ; smaller by one than the number of bytes containing the original packed decimal
                         0538  ; string. One case is a divisor string with an even number of digits. The
                         0538  ; second case is a divisor string with an odd number of digits but the most
                         0538  ; significant digit is zero (essentially a variant of the first case). The
                         0538  ; simplest way to handle all of these cases is to decrement R8, the divisor
                         0538  ; counter, if R2 is zero. Note that previous checks for a zero divisor
                         0538  ; prevent R8 from going to zero.
                         0538
           63   52   D0  0538          MOVL    R2,(R3)                   ; Store final quotient
                0A   12  053B          BNEQ    25$                       ; Leave well enough alone if nonzero
                56   D6  053D          INCL    R6                        ; One more quotient digit
           57   04   C2  053F          SUBL    #4,R7                     ; Make room for it
                58   D7  0542          DECL    R8                        ; Count one less divisor "digit"
                01   12  0544          BNEQ    25$
                         0546
                         0546  ;;;   ********** BEGIN TEMP **********
                         0546  ;;;
                         0546  ;;;   THE FOLLOWING HALT INSTRUCTION SHOULD BE REPLACED WITH THE CORRECT
                         0546  ;;;   ABORT CODE.
                         0546  ;;;
                         0546  ;;;   THE HALT IS SIMILAR TO THE
                         0546  ;;;
```

```
                     0546  ;;;          MICROCODE CANNOT GET HERE
                     0546  ;;;
                     0546  ;;; ERRORS THAT OTHER IMPLEMENTATIONS USE.
                     0546  ;;;
                 00  0546               halt                            ; This will cause an OPCDEC exception
                     0547  ;;;
                     0547  ;;; *********** END TEMP ***********
                     0547
       59   5E   D0  0547  25$:         MOVL    SP,R9                   ; R9 locates low order divisor digit
                     054A
                     054A  ; The dividend is stored on the stack as an array of longwords. It does not
                     054A  ; have its digit pairs shifted so that this storage loop is simpler. An extra
                     054A  ; place is set aside in the event that it is necessary to normalize the
                     054A  ; dividend and divisor before division is attempted.
                     054A
            7E   D4  054A               CLRL    -(SP)                   ; Set aside space for U[0]
     52   6746   DE  054C               MOVAL   (R7)[R6],R2             ; Retrieve dividend descriptor
       52   62   7D  0550               MOVQ    (R2),R2                 ; ... in two steps
                     0553
                     0553               MARK_POINT      DIVP_R6_R7
       51   83   9A  0553  30$:         MOVZBL  (R3)+,R1                ; Get next decimal digit pair
 7E   0000'CF41  9A  0556               MOVZBL  DECIMAL$PACKED_TO_BINARY_TABLE[R1],-
                     055C                       -(SP)                   ; Convert digits to binary
       F4 52     F5  055C               SOBGTR  R2,30$                  ; Loop through entire input string
                     055F
                     055F  ; From this point until the common exit path for MULP and DIVP is entered,
                     055F  ; no access violations that need to be backed out can occur. We do not need
                     055F  ; to keep the address of ARITH_ACCVIO in R10 for this stretch of code. Note
                     055F  ; that R10 must be reloaded before the exit code executes because the
                     055F  ; destination string is written and may cause access violations.
                     055F
     5A   6746   D0  055F               MOVL    (R7)[R6],R10            ; Retrieve size of dividend array
       5B   5E   D0  0563               MOVL    SP,R11                  ; R11 locates low order dividend digit
                     0566
                     0566  ; Allocate a scratch array on the stack the same size as the divisor array
                     0566  ; (which is one larger than the number of digit pairs)
                     0566
       52   58   CE  0566               MNEGL   R8,R2                   ; Need a negative index
 5E   FC AE42    DE  0569               MOVAL   -4(SP)[R2],SP           ; Adjust stack pointer
                     056E
                     056E  ;+
                     056E  ; At this point, the stack and relevant general registers contain the
                     056E  ; following information. In this description, N represents the number
                     056E  ; of digit pairs in the divisor and M represents the number of digit
                     056E  ; pairs in the dividend.
                     056E  ;
                     056E  ;                +--------------------------------+ <-- SP
                     056E  ;    scratch     :         N+1 longwords          :
                     056E  ;                +--------------------------------+ <-- R11
                     056E  ;    dividend    :         M+1 longwords          :
                     056E  ;                +--------------------------------+ <-- R9
                     056E  ;    divisor     :         N+1 longwords          :
                     056E  ;                +--------------------------------+ <-- R7
                     056E  ;    quotient    :        M+1-N longwords         :
                     056E  ;                +--------------------------------+
                     056E  ;                :            R0..R11             :
                     056E  ;                +--------------------------------+
```

```
                          056E  ;
                          056E  ;       R6  - Number of longwords in quotient array (M+1-N)
                          056E  ;       R7  - Address of beginning of quotient array
                          056E  ;       R8  - Number of digit pairs in divisor (called N)
                          056E  ;       R9  - Address of low order digits in divisor
                          056E  ;       R10 - Number of digit pairs in dividend (called M)
                          056E  ;       R11 - Address of low order digits in dividend
                          056E  ;-
                          056E
              6E     DF   056E          PUSHAL  (SP)                    ; Store address of scratch array
        7E    58     7D   0570          MOVQ    R8,-(SP)                ; Remember divisor descriptor
        7E    5A     7D   0573          MOVQ    R10,-(SP)               ; Remember dividend descriptor
                          0576
                          0576  ; The algorithm that guesses the quotient digit can be guaranteed to be off
                          0576  ; by no more than two if the high order digit of the divisor (called V[1]) is
                          0576  ; at least as large as 50 (our radix divided by 2). If the high order digit
                          0576  ; is too small, we "normalize" the numerator and denomonator by multiplying
                          0576  ; them by the same number, namely 100/(V[1]+1).
                          0576
   FC A948    01     C1   0576          ADDL3   #1,-4(R9)[R8],R0        ; Compute V[1] + 1
              50          057B
        33    50     D1   057C          CMPL    R0,#51                  ; Compare to 50 + 1
              14     18   057F          BGEQ    40$                     ; Skip normalization if V[1] big enough
              50     C7   0581          DIVL3   R0,#100,R3              ; Compute normalization factor
   00000064   8F          0583
              53          0588
        54    58     7D   0589          MOVQ    R8,R4                   ; Get descriptor of divisor
            00E0     30   058C          BSBW    MULTIPLY_STRING         ; Normalize divisor
        54    5A     7D   058F          MOVQ    R10,R4                  ; Get descriptor of dividend
            00DA     30   0592          BSBW    MULTIPLY_STRING         ; Normalize dividend
                          0595
                          0595  ; We have now reached the point where we can start calculating quotient digits.
                          0595  ; In the following loop, R5 and R6 are loop invariants. R5 contains the number
                          0595  ; of digit pairs in the divisor. R6 always points to the longword beyond the
                          0595  ; most significant digit in the dividend string. R7 and R8 must be loaded on
                          0595  ; each pass through because these two pointers are modified. Notice that the
                          0595  ; address of the divisor array is exactly what we want to store in R6.
                          0595
        5A    56     7D   0595  40$:    MOVQ    R6,R10                  ; Let R10/R11 describe quotient and loop
              5B     DD   0598          PUSHL   R11                     ; Save quotient address for exit code
   5B       6B4A     DE   059A          MOVAL   (R11)[R10],R11          ; Store quotient digits from high end
                          059E
                          059E  ; This rather harmless looking loop is where the work is done
                          059E
        55    58     7D   059E          MOVQ    R8,R5                   ; Initialize count and dividend address
        59    5A     D0   05A1          MOVL    R10,R9                  ; Remember the loop count in R9
                          05A4
   57      10 AE     7D   05A4  50$:    MOVQ    16(SP),R7               ; Load divisor and scratch addresses
            001F     30   05A8          BSBW    QUOTIENT_DIGIT          ; Get the next quotient digit
        7B    53     D0   05AB          MOVL    R3,-(R11)               ; Store it
        56    04     C2   05AE          SUBL    #4,R6                   ; "Advance" dividend pointer
           F0 5A     F5   05B1          SOBGTR  R10,50$                 ; ... and go gack for more
                          05B4
                          05B4  ; The quotient digits have been stored on the stack. Eliminate the rest of the
                          05B4  ; stack storage and enter the completion code that this routine shares with
                          05B4  ; VAX$MULP. Note that R9 is already set up with the longword count used by
                          05B4  ; the exit code. Note also that R11 is pointing to the saved dividend descriptor
```

```
                    05B4  ; that sits on top of the saved register array.
                    05B4
       5E   6E  D0  05B4          MOVL    (SP),SP                  ; Reset stack pointer
   54  18 AB49  DE  05B7          MOVAL   < <4*2> + -              ; Skip over saved dividend descriptor
                    05BC                  <4*4> >(R11)[R9],R4      ;  and retrieve original R4 and R5
       54   64  7D  05BC          MOVQ    (R4),R4                  ; ... in two steps
                    05BF
                    05BF  ; The following is a HACK.
                    05BF  ;
                    05BF  ; The method used to obtain quotient digits generally leaves garbage (nonzero)
                    05BF  ; in what will become the sign digit. (In fact, this is the tenths digit of a
                    05BF  ; decimal expansion of the remainder.) We need to make the least significant
                    05BF  ; digit a multiple of ten.
                    05BF
   50  6E   0A  C7  05BF          DIVL3   #10,(SP),R0              ; Divide by ten, losing remainder
   6E  50   0A  C5  05C3          MULL3   #10,R0,(SP)              ; Store only tens digit
                    05C7
          FDB3  31  05C7          BRW     MULTIPLY_DIVIDE_EXIT     ; Join common exit code
                    05CA
                    05CA          .DISABLE        LOCAL_BLOCK
```

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

```
                         05CA  ;       quadword result of EMUL that is then passed into EDIV. R2 is the
                         05CA  ;       carry from one step to the next. R4 is the loop counter.
                         05CA  ;-
                         05CA
                         05CA  QUOTIENT_DIGIT:
    00000064 8F     7A   05CA          EMUL    #100,-4(R6),-8(R6),R0   ; R0 <- 100 * U[j] + U[j+1]
    F8 A6    FC A6       05D0
             50          05D4
    50   FC A745    C6   05D5          DIVL2   -4(R7)[R5],R0           ; R0 <- R0 / V[1]
         53   50    D0   05DA          MOVL    R0,R3                   ; Store quotient "digit" in R3
              65    13   05DD          BEQL    45$                     ; Nothing to do if quotient is zero
              53    D1   05DF          CMPL    R3,#100                 ; Is quotient LEQU 99?
    00000064 8F          05E1
              07    1F   05E6          BLSSU   5$                      ; Branch if quotient OK
    00000063 8F     D0   05E8          MOVL    #99,R3                  ; Otherwise start with 99
             53          05EE
                         05EF
                         05EF  ; We will now multiply the divisor array by the quotient digit, storing the
                         05EF  ; product in the scratch array.
                         05EF
              52    D4   05EF  5$:     CLRL    R2                      ; Start out with a carry of zero
         54   55    D0   05F1          MOVL    R5,R4                   ; R4 will be the loop counter
                         05F4
    52   87   53    7A   05F4  10$:    EMUL    R3,(R7)+,R2,R0          ; Multiply next divisor digit
              50         05F8
    00000064 8F     7B   05F9          EDIV    #100,R0,R2,(R8)+        ; Remainder to input array
    88   52   50         05FF
                         0602                                          ; Quotient becomes carry
         EF   54    F5   0602          SOBGTR  R4,10$                  ; More divisor digits?
                         0605
         88   52    D0   0605          MOVL    R2,(R8)+                ; Store final carry
                         0608
                         0608  ; If the product array is larger than the dividend array, then the quotient is
                         0608  ; too large. To avoid a second trip through the rather costly EMUL/EDIV loop,
                         0608  ; and also to avoid array subtraction that produces a negative result, we will
                         0608  ; first compare the product and dividend arrays. If the product is smaller, we
                         0608  ; can safely subtract. If the product is larger, we decrease the quotient by
                         0608  ; one and subtract the divisor array from the product array.
                         0608
         50   56    D0   0608  15$:    MOVL    R6,R0                   ; Point R0 and R1 to high address ends
         51   58    D0   060B          MOVL    R8,R1                   ; ... of dividend and scratch strings
         54   55    D0   060E          MOVL    R5,R4                   ; Initialize the loop counter
                         0611
                         0611  ; The comparison is done from most to least significant digits
                         0611
         70   71    D1   0611  20$:    CMPL    -(R1),-(R0)             ; Compare next pair of digits
              0E    1F   0614          BLSSU   30$                     ; Leave loop if product is smaller
              2D    1A   0616          BGTRU   50$                     ; Also leave if product is larger
         F6   54    F4   0618          SOBGEQ  R4,20$                  ; More to test?
                         061B
                         061B  ; If we drop through the loop, then the dividend and product are equal. We
                         061B  ; simply store zeros in the dividend array (the equivalent of subtraction
                         061B  ; of equal arrays) and return. Note that R0 is already pointing to the
                         061B  ; least significant dividend array element.
                         061B
         54   55    D0   061B          MOVL    R5,R4                   ; Initialize still another loop counter
                         061E
```

```
            80   D4  061E  25$:    CLRL    (R0)+                   ; Store another zero
         FB 54   F4  0620          SOBGEQ  R4,25$                  ; Keep going?
                     0623
            05       0623          RSB                             ; Return to caller
                     0624
                     0624  ; If we drop through the loop, then the quotient that is stored in R3 is good.
                     0624  ; We need to subtract the product array from the dividend array. Note that R0
                     0624  ; and R1 need to be adjusted to point to the least significant array elements
                     0624  ; before the subtraction can begin.
                     0624
      54    54   CE  0624  30$:    MNEGL   R4,R4                   ; We need a negative index
    50    6044   DE  0627          MOVAL   (R0)[R4],R0             ; Adjust dividend pointer
    51    6144   DE  062B          MOVAL   (R1)[R4],R1             ; ... and product pointer
      54    55   D0  062F          MOVL    R5,R4                   ; R4 will count still another loop
                     0632
      80    81   C2  0632  35$:    SUBL2   (R1)+,(R0)+             ; Subtract next digits
            0A   18  0635          BGEQ    40$                     ; Skip to end of loop if no borrow
   00000064 8F   C0  0637          ADDL2   #100,-4(R0)             ; Add borrow back to this digit
         FC A0       063D
            60   D7  063F          DECL    (R0)                    ; ... and borrow from next highest digit
         EE 54   F4  0641  40$:    SOBGEQ  R4,35$                  ; Keep going?
                     0644
                     0644  ; This is the exit path. R3 contains the quotient digit. The pointers to the
                     0644  ; various input and scratch arrays are in an indeterminate state.
                     0644
            05       0644  45$:    RSB                             ; Return to caller
                     0645
                     0645  ; The first guess at the quotient digit is too large. The brute force
                     0645  ; approach is to decrement the quotient by one and execute the EMUL/EDIV loop
                     0645  ; again. Note, however, that we can evaluate the modified product by
                     0645  ; subtracting the divisor from the initial product. Note also that, because
                     0645  ; the leading digit in the divisor is "large enough", we can only end up in
                     0645  ; this code path twice. (That is, the initial guess at the quotient will
                     0645  ; never be off by more than two.)
                     0645
            53   D7  0645  50$:    DECL    R3                      ; Try quotient smaller by one
            FB   13  0647          BEQL    45$                     ; All done if zero
                     0649
                     0649  ; Point R1 and R2 at the least significant digits of the scratch and product
                     0649  ; strings respectively.
                     0649
      50    55   CE  0649          MNEGL   R5,R0                   ; Need a negative index
  51   FC A840   DE  064C          MOVAL   -4(R8)[R0],R1           ; Scratch array contains N+1 elements
    52    6740   DE  0651          MOVAL   (R7)[R0],R2             ; Product array contains N elements
      54    55   D0  0655          MOVL    R5,R4                   , R4 will count still another loop
                     0658
      81    82   C2  0658  60$:    SUBL2   (R2)+,(R1)+             ; Subtract next digits
            0A   18  065B          BGEQ    70$                     ; Skip to end of loop if no borrow
   00000064 8F   C0  065D          ADDL2   #100,-4(R1)             ; Add borrow back to this digit
         FC A1       0663
            61   D7  0665          DECL    (R1)                    ; ... and borrow from next highest digit
         EE 54   F4  0667  70$:    SOBGEQ  R4,60$                  ; Keep going?
                     066A
      51    04   C0  066A          ADDL2   #4,R1                   ; Point R1 at most significant digit
            99   11  066D          BRB     15$                     ; Make another comparison
```

```
                           066F        .SUBTITLE       MULTIPLY_STRING - Multiply a String by a Number
                           066F  ;+
                           066F  ; Functional Description:
                           066F  ;
                           066F  ;       This routine multiplies an array of numbers (each array element LEQU
                           066F  ;       99) by a number (also LEQU 99). Each array element in the input array
                           066F  ;       is replaced with the modified product, with the carry propogated to
                           066F  ;       the next array element.
                           066F  ;
                           066F  ; Input Parameters:
                           066F  ;
                           066F  ;       R3 - Multiplier
                           066F  ;       R4 - Input array size
                           066F  ;       R5 - Input array address
                           066F  ;
                           066F  ; Output Parameters:
                           066F  ;
                           066F  ;       None
                           066F  ;
                           066F  ; Implicit Output:
                           066F  ;
                           066F  ;       The input array elements are altered.
                           066F  ;
                           066F  ; Side Effects:
                           066F  ;
                           066F  ;       R4 and R5 are modified by this routine.
                           066F  ;
                           066F  ;       R3 is preserved.
                           066F  ;
                           066F  ;       R0, R1, and R2 are used as scratch registers. R0 and R1 contain the
                           066F  ;       quadword result of EMUL that is then passed into EDIV. R2 is the
                           066F  ;       carry from one step to the next.
                           066F  ;
                           066F  ; Assumptions:
                           066F  ;
                           066F  ;       This routine assumes that all array elements lie in the range from 0
                           066F  ;       to 99 inclusive. (This is true if all input strings contain only legal
                           066F  ;       decimal digits.) The arithmetic performed by this routine will
                           066F  ;       maintain this assumption. The details of this argument can be found in
                           066F  ;       the routine header for EXTENDED_MULTIPLY_STRING. This routine performs
                           066F  ;       less work so that those arguments also apply here.
                           066F  ;-
                           066F
                           066F  MULTIPLY_STRING:
               52     D4   066F          CLRL    R2                      ; Initial carry is zero
                           0671
      52   65  53     7A   0671  10$:    EMUL    R3,(R5),R2,R0           ; Form modified product (R0 LEQU 9900)
               50          0675
   00000064 8F        7B   0676          EDIV    #100,R0,R2,(R5)+        ; Remainder to input array
   85   52     50          067C
                           067F                                          ; Quotient becomes carry
           EF  54     F5   067F          SOBGTR  R4,10$                  ; Keep going?
                           0682
       65      52     D0   0682          MOVL    R2,(R5)                 ; Store final carry
                      05   0685          RSB
```

```
                    0686        .SUBTITLE       DECIMAL_ROPRAND
                    0686 ;-
                    0686 ; Functional Description:
                    0686 ;
                    0686 ;      This routine receives control when a digit count larger than 31
                    0686 ;      is detected. The exception is architecturally defined as an
                    0686 ;      abort so there is no need to store intermediate state. All of the
                    0686 ;      routines in this module save all registers R0 through R11 before
                    0686 ;      performing the digit check. These registers must be restored
                    0686 ;      before control is passed to VAX$ROPRAND.
                    0686 ;
                    0686 ; Input Parameters:
                    0686 ;
                    0686 ;      00(SP) - Saved R0
                    0686 ;        .
                    0686 ;        .
                    0686 ;      44(SP) - Saved R11
                    0686 ;      48(SP) - Return PC from VAX$xxxxxx routine
                    0686 ;
                    0686 ; Output Parameters:
                    0686 ;
                    0686 ;      00(SP) - Offset in packed register array to delta PC byte
                    0686 ;      04(SP) - Return PC from VAX$xxxxxx routine
                    0686 ;
                    0686 ; Implicit Output:
                    0686 ;
                    0686 ;      This routine passes control to VAX$ROPRAND where further
                    0686 ;      exception processing takes place.
                    0686 ;-
                    0686
                    0686        ASSUME ADDP6_B_DELTA_PC EQ ADDP4_B_DELTA_PC
                    0686        ASSUME SUBP4_B_DELTA_PC EQ ADDP4_B_DELTA_PC
                    0686        ASSUME SUBP6_B_DELTA_PC EQ ADDP4_B_DELTA_PC
                    0686        ASSUME MULP_B_DELTA_PC  EQ ADDP4_B_DELTA_PC
                    0686        ASSUME DIVP_B_DELTA_PC  EQ ADDP4_B_DELTA_PC
                    0686
                    0686 DECIMAL_ROPRAND:
OFFF 8F     BA      0686        POPR    #^M<R0,R1,R2,R3,R4,R5,R6,R7,R8,R9,R10,R11>
     03     DD      068A        PUSHL   #ADDP4_B_DELTA_PC       ; Store offset to delta PC byte
  F971'     31      068C        BRW     VAX$ROPRAND             ; Pass control along
```

```
                            068F          .SUBTITLE       ARITH_ACCVIO - Reflect an Access Violation
                            068F  ;+
                            068F  ; Functional Description:
                            068F  ;
                            068F  ;       This routine receives control when an access violation occurs while
                            068F  ;       executing within the emulator routines for ADDP4, ADDP6, SUBP4, SUBP6,
                            068F  ;       MULP, or DIVP.
                            068F  ;
                            068F  ;       The routine header for ASHP_ACCVIO in module VAX$ASHP contains a
                            068F  ;       detailed description of access violation handling for the decimal
                            068F  ;       string instructions.
                            068F  ;
                            068F  ; Input Parameters:
                            068F  ;
                            068F  ;       See routine ASHP_ACCVIO in module VAX$ASHP
                            068F  ;
                            068F  ; Output Parameters:
                            068F  ;
                            068F  ;       See routine ASHP_ACCVIO in module VAX$ASHP
                            068F  ;-
                            068F
                            068F  ARITH_ACCVIO:
                  52   D4   068F          CLRL    R2                      ; Initialize the counter
             F96B CF   9F   0691          PUSHAB  MODULE_BASE             ; Store base address of this module
            06FC'CF    9F   0695          PUSHAB  MODULE_END              ; Store module end address
               F964'   30   0699          BSBW    DECIMAL$BOUNDS_CHECK    ; Check if PC is inside the module
              5E  04   C0   069C          ADDL    #4,SP                   ; Discard end address
              51  8E   C2   069F          SUBL2   (SP)+,R1                ; Get PC relative to this base
                            06A2
      0000'CF42   51   B1   06A2  10$:    CMPW    R1,PC_TABLE_BASE[R2]    ; Is this the right PC?
                  07   13   06A8          BEQL    30$                     ; Exit loop if true
           F4 52  2A   F2   06AA          AOBLSS  #TABLE_SIZE,R2,10$      ; Do the entire table
                            06AE
                            06AE  ; If we drop through the dispatching based on PC, then the exception is not
                            06AE  ; one that we want to back up. We simply reflect the exception to the user.
                            06AE
                  0F   BA   06AE  20$:    POPR    #^M<R0,R1,R2,R3>        ; Restore saved registers
                       05   06B0          RSB                             ; Return to exception dispatcher
                            06B1
                            06B1  ; The exception PC matched one of the entries in our PC table. R2 contains
                            06B1  ; the index into both the PC table and the handler table. R1 has served
                            06B1  ; its purpose and can be used as a scratch register.
                            06B1
   51     0000'CF42    3C   06B1  30$:    MOVZWL  HANDLER_TABLE_BASE[R2],R1       ; Get the offset to the handler
          F944 CF41    17   06B7          JMP     MODULE_BASE[R1]         , Pass control to the handler
                            06BC
                            06BC  ; In all of the instruction-specific routines, the state of the stack
                            06BC  ; will be shown as it was when the exception occurred. All offsets will
                            06BC  ; be pictured relative to R0.
```

```
                06BC        .SUBTITLE       Access Violation Handling for ADDPx and SUBPx
                06BC  ;+
                06BC  ; Functional Description:
                06BC  ;
                06BC  ;       The only difference among the various entry points is the number of
                06BC  ;       longwords on the stack. R0 is advanced beyond these longwords to point
                06BC  ;       to the list of saved registers. These registers are then restored,
                06BC  ;       effectively backing the routine up to its initial state.
                06BC  ;
                06BC  ; Input Parameters:
                06BC  ;
                06BC  ;       R0 - Address of top of stack when access violation occurred
                06BC  ;
                06BC  ;       See specific entry points for details
                06BC  ;
                06BC  ; Output Parameters:
                06BC  ;
                06BC  ;       See input parameter list for VAX$DECIMAL_ACCVIO in module VAX$ASHP
                06BC  ;-
                06BC
                06BC  ;+
                06BC  ; ADD_SUB_BSBW_24
                06BC  ;
                06BC  ; An access violation occurred in one of the subroutines ADD_PACKED_BYTE,
                06BC  ; SUB_PACKED_BYTE, or STORE_RESULT. In addition to the six longwords of work
                06BC  ; space, this routine has an additional longword, the return PC, on the
                06BC  ; stack.
                06BC  ;
                06BC  ;       00(R0) - Return PC in mainline VAX$xxxxxx routine
                06BC  ;       04(R0) - Address of sign byte of destination string
                06BC  ;       08(R0) - First longword of scratch space
                06BC  ;        etc.
                06BC  ;-
                06BC
                06BC  ADD_SUB_BSBW_24:
 50   04   C0   06BC          ADDL    #4,R0                   ; Skip over return PC and drop into ...
                06BF
                06BF  ;+
                06BF  ; ADD_SUB_24
                06BF  ;
                06BF  ; There are five longwords of workspace and a saved string address on the stack
                06BF  ; for this entry point.
                06BF  ;
                06BF  ;       00(R0) - Address of sign byte of destination string
                06BF  ;       04(R0) - First longword of scratch space
                06BF  ;         .
                06BF  ;         .
                06BF  ;       20(R0) - Fifth longword of scratch space
                06BF  ;       24(SP) - Saved R0
                06BF  ;       28(SP) - Saved R1
                06BF  ;        etc.
                06BF  ;-
                06BF
                06BF  ADD_SUB_24:
 50   18   C0   06BF          ADDL    #24,R0                  ; Discard scratch space on stack
    F93B'  31   06C2          BRW     VAX$DECIMAL_ACCVIO      ; Join common code to restore registers
                06C5
```

```
                          06C5  ;+
;LJK0045                  06C5  ; ADD_SUB_BSBW_4
;LJK0045                  06C5  ;
;LJK0045                  06C5  ; An access violation occurred in the subroutine CHECK_WRITE_ACCESS. This
;LJK0045                  06C5  ; entry point has two additional longwords, a copy of the saved R4 and the
;LJK0045                  06C5  ; return PC, on the stack on top of the saved register array.
;LJK0045                  06C5  ;
;LJK0045                  06C5  ;        00(R0) - Saved R4
;LJK0045                  06C5  ;        04(R0) - Return PC in mainline VAX$xxxP4 routine
;LJK0045                  06C5  ;        08(R0) - Saved R0
;LJK0045                  06C5  ;        12(R0) - Saved R1
;LJK0045                  06C5  ;         etc.
;LJK0045                  06C5  ;-
;LJK0045                  06C5
;LJK0045                  06C5  ADD_SUB_BSBW_4:
;LJK0045     50  04  C0   06C5          ADDL    #4,R0                   ; Skip over saved R4 and drop into ...
;LJK0045                  06C8
;LJK0045                  06C8  ;+
                          06C8  ; ADD_SUB_BSBW_0
                          06C8  ;
                          06C8  ; An access violation occurred in one of the subroutine STRIP_ZEROS. This
                          06C8  ; entry point has an additional longword, the return PC, on the stack on top
                          06C8  ; of the saved register array.
                          06C8  ;
                          06C8  ;        00(R0) - Return PC in mainline VAX$xxxxxx routine
                          06C8  ;        04(R0) - Saved R0
                          06C8  ;        08(R0) - Saved R1
                          06C8  ;         etc.
                          06C8  ;-
                          06C8
                          06C8  ADD_SUB_BSBW_0:
             50  04  C0   06C8          ADDL    #4,R0                   ; Skip over return PC and ...
               F932'  31  06CB          BRW     VAX$DECIMAL_ACCVIO      ; Join common code to restore registers
```

```
                      06CE          .SUBTITLE       Access Violation Handling for MULP and DIVP
                      06CE  ;+
                      06CE  ; Functional Description:
                      06CE  ;
                      06CE  ;       The only difference among the various entry points is the number of
                      06CE  ;       longwords on the stack. R0 is advanced beyond these longwords to point
                      06CE  ;       to the list of saved registers. These registers are then restored,
                      06CE  ;       effectively backing the routine up to its initial state.
                      06CE  ;
                      06CE  ; Input Parameters:
                      06CE  ;
                      06CE  ;       R0 - Address of top of stack when access violation occurred
                      06CE  ;
                      06CE  ;       See specific entry points for details
                      06CE  ;
                      06CE  ; Output Parameters:
                      06CE  ;
                      06CE  ;       See input parameter list for VAX$DECIMAL_ACCVIO in module VAX$ASHP
                      06CE  ;-
                      06CE
                      06CE  ;+
                      06CE  ; MULP_R8
                      06CE  ;
                      06CE  ; An access violation occurred while MULP was accessing one of its two source
                      06CE  ; strings. In this particular case, MULP was storing the longer of the two
                      06CE  ; input strings in a longword array on the top of the stack. There is an
                      06CE  ; array of R8 longwords on top of an array of 32 longwords on top of the
                      06CE  ; saved register array.
                      06CE  ;
                      06CE  ;       R8 - Number of longwords on top of the 32-longword array
                      06CE  ;-
                      06CE
                      06CE          .ENABLE         LOCAL_BLOCK
                      06CE
                      06CE  MULP_R8:
    50    6048   DE   06CE          MOVAL   (R0)[R8],R0             ; Discard input array storage
            04   11   06D2          BRB     10$                     ; Might as well share a little code
                      06D4
                      06D4  ;+
                      06D4  ; MULP_AT_SP
                      06D4  ;
                      06D4  ; An access violation occurred while MULP was accessing one of its two source
                      06D4  ; strings. In this case, the access violation occurred in the middle of the
                      06D4  ; grand multiply loop as a digit pair was being retrieved from the shorter of
                      06D4  ; the two input strings. The address of the start of the 32-longword array
                      06D4  ; was itself stored on top of the stack for convenience.
                      06D4  ;
                      06D4  ;       00(R0) - Saved byte count of longer input string
                      06D4  ;       04(R0) - Saved address of longer input string
                      06D4  ;       08(R0) - Address of 32-longword array farther down the stack
                      06D4  ;-
                      06D4
                      06D4  MULP_AT_SP:
    50   08 A0   D0   06D4          MOVL    8(R0),R0                ; Locate start of 32-longword array
  50  0088 C0    9E   06D8  10$:    MOVAB   <<4*32> + <4*2>>(R0),R0 ; Throw that away, too
         F920'   31   06DD          BRW     VAX$DECIMAL_ACCVIO      ; Join common code to restore registers
                      06E0
```

```
                      06E0          .DISABLE        LOCAL_BLOCK
                      06E0
                      06E0  ;+
                      06E0  ; MULP_DIVP_R9
                      06E0  ;
                      06E0  ; An access violation occurred while the final result was being stored in the
                      06E0  ; result string. In this common exit code path, R9 counts the number of
                      06E0  ; longwords on the stack. In all cases where an access violation can occur, a
                      06E0  ; longword has been removed from the stack but R9 has not yet been
                      06E0  ; decremented to reflect this. The conceptual instruction sequence that
                      06E0  ; resets the stack pointer (really R0) to point to the start of the saved
                      06E0  ; register array is
                      06E0  ;
                      06E0  ;       DECL    R9
                      06E0  ;       MOVAL   (R0)[R9]
                      06E0  ;
                      06E0  ; A single instruction accomplishes this.
                      06E0  ;
                      06E0  ;       R9 - One more than the number of longwords on the stack on top
                      06E0  ;               of the saved register array.
                      06E0  ;
                      06E0  ;       00(R0) - First longword of scratch storage remaining on the stack
                      06E0  ;          .
                      06E0  ;          .
                      06E0  ;       zz-4(R0)  - Last longword of scratch storage
                      06E0  ;       zz+0(R0)  - Saved count of dividend or multiplier string
                      06E0  ;       zz+4(R0)  - Saved address of dividend or multiplier string
                      06E0  ;       zz+8(R0)  - Saved R0
                      06E0  ;       zz+12(R0) - Saved R1
                      06E0  ;        etc.
                      06E0  ;
                      06E0  ;       where zz = 4 * (R9 - 1)
                      06E0  ;-
                      06E0
                      06E0  MULP_DIVP_R9:
     50   04 A049  DE 06E0          MOVAL   4(R0)[R9],R0            ; Discard scratch storage on stack
             F918' 31 06E5          BRW     VAX$DECIMAL_ACCVIO      ; Join common code to restore registers
                      06E8
                      06E8  ;+
                      06E8  ; MULP_DIVP_8
                      06E8  ;
                      06E8  ; An access violation occurred in the common exit path after the scratch array
                      06E8  ; had been removed from the stack but before the saved descriptor for the
                      06E8  ; multiplier string was discarded.
                      06E8  ;
                      06E8  ;       0(R0)  - Saved count of dividend or multiplier string
                      06E8  ;       4(R0)  - Saved address of dividend or multiplier string
                      06E8  ;       8(R0)  - Saved R0
                      06E8  ;       12(R0) - Saved R1
                      06E8  ;        etc.
                      06E8  ;-
                      06E8
                      06E8  MULP_DIVP_8:
     50   08       C0 06E8          ADDL    #8,R0                   ; Discard multiplier string descriptor
             F912' 31 06EB          BRW     VAX$DECIMAL_ACCVIO      ; Join common code to restore registers
                      06EE
                      06EE  ;+
```

```
                 06EE  ; MULP_BSBW_0
                 06EE  ; DIVP_BSBW_0
                 06EE  ;
                 06EE  ; An access violation occurred in one of the subroutine STRIP_ZEROS. This
                 06EE  ; entry point has an additional longword, the return PC, on the stack on top
                 06EE  ; of the saved register array.
                 06EE  ;
                 06EE  ;         00(R0) - Return PC in mainline VAX$MULP or VAX$DIVP routine
                 06EE  ;         04(R0) - Saved R0
                 06EE  ;         08(R0) - Saved R1
                 06EE  ;          etc.
                 06EE  ;-
                 06EE
                 06EE  MULP_BSBW_0:
                 06EE  DIVP_BSBW_0:
   50   04   C0  06EE          ADDL    #4,R0                   ; Skip over return PC and drop into ...
                 06F1
                 06F1  ;+
                 06F1  ; DIVP_0
                 06F1  ; MULP_DIVP_0
                 06F1  ;
                 06F1  ; There was nothing allocated on the stack other than the saved register
                 06F1  ; array when the access violation occurred. We merely pass control to common
                 06F1  ; code to restore the registers.
                 06F1  ;
                 06F1  ;         00(R0) - Saved R0
                 06F1  ;         04(R0) - Saved R1
                 06F1  ;          etc.
                 06F1  ;-
                 06F1
                 06F1  DIVP_0:
                 06F1  MULP_DIVP_0:
      F90C'  31  06F1          BRW     VAX$DECIMAL_ACCVIO      ; Join common code to restore registers
                 06F4
                 06F4  ;+
                 06F4  ; DIVP_R6_R7
                 06F4  ;
                 06F4  ; An access violation occurred while one of the two input strings was being
                 06F4  ; converted to an array of longwords on the stack. The state of the stack
                 06F4  ; is rather complicated but R6 and R7 contain enough information to allow
                 06F4  ; the rest of the stack contents to be ignored.
                 06F4  ;
                 06F4  ;         R6 - Count of longwords in quotient array on stack
                 06F4  ;         R7 - Address of quotient array on stack
                 06F4  ;
                 06F4  ;         00(R0) - First longword of quotient array
                 06F4  ;            .
                 06F4  ;            .
                 06F4  ;         zz-4(R0) - Last longword of scratch storage
                 06F4  ;         zz+0(R0) - Digit count of dividend string
                 06F4  ;         zz+4(R0) - Address of dividend string
                 06F4  ;         zz+8(R0) - Saved R0
                 06F4  ;         zz+12(R0) - Saved R1
                 06F4  ;          etc.
                 06F4  ;
                 06F4  ;         where zz = 4 * R6
                 06F4  ;-
```

```
                          06F4
                          06F4   DIVP_R6_R7:
         50  08 A746  DE  06F4           MOVAL   8(R7)[R6],R0            ; Discard everything on stack
                F904' 31  06F9           BRW     VAX$DECIMAL_ACCVIO      ; Join common code to restore registers
                          06FC
                          06FC           END_MARK_POINT
                          06FC
                          06FC           .END
```

```
...PC...                          = 00000553
...ROPRAND...                     = 000004BF R        02
ADDP4_B_DELTA_PC                  = 00000003
ADDP6_B_DELTA_PC                  = 00000003
ADD_PACKED                          000000E5 R        02
ADD_PACKED_BYTE_R6_R7               00000170 R        02
ADD_PACKED_BYTE_STRING              0000016A R        02
ADD_SUBTRACT_EXIT                   0000013D R        02
ADD_SUB_24                          000006BF R        02
ADD_SUB_BSBW_0                      000006C8 R        02
ADD_SUB_BSBW_24                     000006BC R        02
ADD_SUB_BSBW_4                      000006C5 R        02
ADD_SUB_V_ZERO_R4                 = 0000001F
ARITH_ACCVIO                        0000068F R        02
CHECK_WRITE_ACCESS                  00000292 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$STRIP_ZEROS_R2_R3           ********    X     00
DECIMAL_ROPRAND                     00000686 R        02
DIVIDE_BY_ZERO                      0000049E R        02
DIVP_0                              000006F1 R        02
DIVP_BSBW_0                         000006EE R        02
DIVP_B_DELTA_PC                   = 00000003
DIVP_R6_R7                          000006F4 R        02
EXTEND_STRING_MULTIPLY              00000470 R        02
HANDLER_TABLE_BASE                  00000000 R        04
MODULE_BASE                       = 00000000 R        02
MODULE_END                        = 000006FC R        02
MULP_AT_SP                          000006D4 R        02
MULP_BSBW_0                         000006EE R        02
MULP_B_DELTA_PC                   = 00000003
MULP_DIVP_0                         000006F1 R        02
MULP_DIVP_8                         000006E8 R        02
MULP_DIVP_R9                        000006E0 R        02
MULP_R8                             000006CE R        02
MULTIPLY_DIVIDE_EXIT                0000037D R        02
MULTIPLY_STRING                     0000066F R        02
PC_TABLE_BASE                       00000000 R        03
PSL$M_N                           = 00000008
PSL$M_V                           = 00000002
PSL$M_Z                           = 00000004
PSL$V_CM                          = 0000001F
PSL$V_V                           = 00000001
PSL$V_Z                           = 00000002
QUOTIENT_DIGIT                      000005CA R        02
SRM$K_FLT_DIV_T                   = 00000004
STORE_RESULT                        00000254 R        02
SUBP4_B_DELTA_PC                  = 00000003
SUBP6_B_DELTA_PC                  = 00000003
SUBTRACT_PACKED                     00000198 R        02
SUB_PACKED_BYTE_R6_R7               0000022E R        02
SUB_PACKED_BYTE_STRING              00000228 R        02
TABLE_SIZE                        = 0000002A
VAX$ADDP4                           00000030 RG       02
VAX$ADDP6                           00000009 RG       02
VAX$ADD_PACKED_BYTE_R6_R7           00000170 RG       02
VAX$DECIMAL_ACCVIO                  ********    X     00
VAX$DECIMAL_EXIT                    ********    X     00
VAX$DIVP                            000004B1 RG       02
VAX$MULP                            000002AD RG       02
VAX$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                 000006FC  ( 1788.)  02 (  2.)    PIC  USR  CON  REL  LCL   SHR   EXE   RD  NOWRT NOVEC LONG
PC_TABLE                  00000054  (   84.)  03 (  3.)    PIC  USR  CON  REL  LCL   SHR NOEXE   RD  NOWRT NOVEC BYTE
HANDLER_TABLE             00000054  (   84.)  04 (  4.)    PIC  USR  CON  REL  LCL   SHR NOEXE   RD  NOWRT NOVEC BYTE

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

Phase                    Page faults   CPU Time      Elapsed Time
-----                    -----------   --------      ------------
Initialization                    75   00:00:00.20   00:00:02.24
Command processing                79   00:00:00.46   00:00:03.73
Pass 1                           241   00:00:09.83   00:00:40.27
Symbol table sort                  0   00:00:00.42   00:00:02.26
Pass 2                           393   00:00:06.36   00:00:32.16
Symbol table output                0   00:00:00.08   00:00:00.54
Psect synopsis output              0   00:00:00.04   00:00:00.36
Cross-reference output             0   00:00:00.00   00:00:00.00
Assembler run totals             788   00:00:17.39   00:01:21.56

The working set limit was 1800 pages.
52274 bytes (103 pages) of virtual memory were used to buffer the intermediate code.
There were 20 pages of symbol table space allocated to hold 184 non-local and 116 local symbols.
2646 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
------------------                            --------------
_$255$DUA18:[EMULAT.OBJ]VAXMACROS.MLB;1                   12
_$255$DUA18:[SYSLIB]STARLET.MLB;3                          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$:VAXARITH/UPDATE=(BUG$:VAXARITH)+LIB$:VAXMACROS/LIB
```

0441 AH-EF71A-SE
VAX/VMS V4.1 SRC LST MCRF UPD

VAXARITH
LIS

EMULAT

ERFPROC1
MAP

VAXEMUL
MAP

ERF

RECSELECT
LIS

ERF
MAP