```
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  UUUUUUUUUUUUUUUU  LLLLLLLLLLLLLLLL AAA     AAA         TTT
EEEEEEEEEEEEEEE   MMM       MMM  UUUUUUUUUUUUUUUU  LLLLLLLLLLLLLLLL AAA     AAA         TTT
EEEEEEEEEEEEEEE   MMM       MMM  UUUUUUUUUUUUUUUU  LLLLLLLLLLLLLLLL AAA     AAA         TTT
```

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

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

Table of contents

```
                        .TITLE  VAX$DECIMAL_ARITHMETIC - VAX-11 Packed Decimal Arithmetic Instructio
:LJK0045                .IDENT  /V04-001/
-i
                ;
                ;*****************************************************************************
                ;*                                                                           *
                ;*  COPYRIGHT (c) 1978, 1980, 1982, 1984 BY                                  *
                ;*  DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.                   *
                ;*  ALL RIGHTS RESERVED.                                                     *
                ;*                                                                           *
                ;*  THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED    *
                ;*  ONLY IN  ACCORDANCE WITH  THE  TERMS  OF  SUCH  LICENSE  AND WITH THE    *
                ;*  INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR  ANY  OTHER     *
                ;*  COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY    *
                ;*  OTHER PERSON.  NO TITLE TO AND OWNERSHIP OF  THE  SOFTWARE IS  HEREBY    *
                ;*  TRANSFERRED.                                                             *
                ;*                                                                           *
                ;*  THE INFORMATION IN THIS SOFTWARE IS  SUBJECT TO CHANGE WITHOUT NOTICE    *
                ;*  AND  SHOULD  NOT  BE  CONSTRUED AS  A COMMITMENT BY DIGITAL EQUIPMENT    *
                ;*  CORPORATION.                                                             *
                ;*                                                                           *
                ;*  DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE  OR  RELIABILITY OF ITS    *
                ;*  SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.                  *
                ;*                                                                           *
                ;*                                                                           *
                ;*****************************************************************************
                ;
                
                ;++
                ; Facility:
                ;
                ;       VAX-11 Instruction Emulator
                ;
                ; Abstract:
                ;
                ;       The routines in this module emulate the VAX-11 packed decimal
                ;       instructions that perform arithmetic operations. These procedures can
                ;       be a part of an emulator package or can be called directly after the
                ;       input parameters have been loaded into the architectural registers.
                ;
                ;       The input parameters to these routines are the registers that
                ;       contain the intermediate instruction state.
                ;
                ; Environment:
                ;
                ;       These routines run at any access mode, at any IPL, and are AST
                ;       reentrant.
                ;
                ; Author:
                ;
                ;       Lawrence J. Kenah
                ;
                ; Creation Date
                ;
                ;       19 October 1983
                ;
                ; Modified by:
```

```
                     0000     58 ;
                     0000     .1 ;       V04-001 LJK0045         Lawrence J. Kenah       19-Sep-1984
                     0000     .2 ;               The result string in ADDP4 and SUBP4 must be probed for
                     0000     .3 ;               write access to insure that the instruction does not modify
                     0000     .4 ;               memory before trying to restart.
                     0000     .5 ;
                     0000     59 ;       V01-003 LJK0037         Lawrence J. Kenah       17-Jul-1984
                     0000     60 ;               Fix two minor bugs in exception handling code that caused
                     0000     61 ;               MULP and DIVP tests to generate spurious access violations.
                     0000     62 ;
                     0000     63 ;       V01-002 LJK0024         Lawrence J. Kenah       21-Feb-1984
                     0000     64 ;               Add code to handle access violations. Perform minor cleanup.
                     0000     65 ;               Eliminate double use of R10 in MULP and DIVP.
                     0000     66 ;
                     0000     67 ;       V01-001 LJK0008         Lawrence J. Kenah       19-Oct-1983
                     0000     68 ;               The emulation code for ADDP4, ADDP6, SUBP4, SUBP6, MULP and
                     0000     69 ;               DIVP was moved into a separate module.
                     0000     70 ;--
```

```
                     0000     72          .SUBTITLE       Declarations
                     0000     73
                     0000     74 ; Include files:
                     0000     75
                     0000     76          .NOCROSS                        ; No cross reference for these
                     0000     77          .ENABLE         SUPPRESSION     ; No symbol table entries either
                     0000     78
                     0000     79          ADDP4_DEF                       ; Bit fields in ADDP4 registers
                     0000     80          ADDP6_DEF                       ; Bit fields in ADDP6 registers
                     0000     81          DIVP_DEF                        ; Bit fields in DIVP registers
                     0000     82          MULP_DEF                        ; Bit fields in MULP registers
                     0000     83          SUBP4_DEF                       ; Bit fields in SUBP4 registers
                     0000     84          SUBP6_DEF                       ; Bit fields in SUBP6 registers
                     0000     85
                     0000     86          $PSLDEF                         ; Define bit fields in PSL
                     0000     87          $SRMDEF                         ; Define arithmetic trap codes
                     0000     88
                     0000     89          .DISABLE        SUPPRESSION     ; Turn on symbol table again
                     0000     90          .CROSS                          ; Cross reference is OK now
                     0000     91
                     0000     92 ; Symbol definitions
                     0000     93
                     0000     94 ;            The architecture requires that R4 be zero on completion of an ADDP6 or
                     0000     95 ;            SUBP6 instruction. If we did not have to worry about restarting
                     0000     96 ;            instructions after an access violation, we could simply zero the saved
                     0000     97 ;            R4 value on the code path that these two instructions have in common
                     0000     98 ;            before they merge with the ADDP4 and SUBP4 routines. The ability to
                     0000     99 ;            restart requires that we keep the original R4 around at least until no
                     0000    100 ;            more access violations are possible. To accomplish this, we store the
                     0000    101 ;            fact that R4 must be cleared on exit in R11, which also contains the
                     0000    102 ;            evolving condition codes. We use bit 31, the compatibility mode bit
                     0000    103 ;            because it is nearly impossible to enter the emulator with CM set.
                     0000    104
                     0000    105          ADD_SUB_V_ZERO_R4 = PSL$V_CM
                     0000    106
                     0000    107 ; External declarations
                     0000    108
                     0000    109          .DISABLE        GLOBAL
                     0000    110
                     0000    111          .EXTERNAL -
                     0000    112                          DECIMAL$BOUNDS_CHECK,-
                     0000    113                          DECIMAL$BINARY_TO_PACKED_TABLE,-
                     0000    114                          DECIMAL$PACKED_TO_BINARY_TABLE,-
                     0000    115                          DECIMAL$STRIP_ZEROS_R0_R1,-
                     0000    116                          DECIMAL$STRIP_ZEROS_R2_R3
                     0000    117
                     0000    118          .EXTERNAL -
                     0000    119                          VAX$DECIMAL_EXIT,-
                     0000    120                          VAX$DECIMAL_ACCVIO,-
                     0000    121                          VAX$REFLECT_TRAP,-
                     0000    122                          VAX$ROPRAND
                     0000    123
                     0000    124 ; PSECT Declarations:
                     0000    125
                     0000    126          .DEFAULT        DISPLACEMENT , WORD
                     0000    127
                 00000000    128          .PSECT _VAX$CODE PIC, USR, CON, REL, LCL, SHR, EXE, RD, NOWRT, LONG
```

```
                                                H 7
0000   129
0000   130          BEGIN_MARK_POINT
```

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

```
                                0009   184          .SUBTITLE       VAX$ADDP6 - Add Packed (6 Operand Format)
                                0009   185 ;+
                                0009   186 ; Functional Description:
                                0009   187 ;
                                0009   188 ;       In 6 operand format, the addend 1  string  specified  by  the  addend  1
                                0009   189 ;       length  and  addend  1  address operands is added to the addend 2 string
                                0009   190 ;       specified by the addend 2 length and addend 2 address operands.  The sum
                                0009   191 ;       string  specified by the sum length and sum address operands is replaced
                                0009   192 ;       by the result.
                                0009   193 ;
                                0009   194 ; Input Parameters:
                                0009   195 ;
                                0009   196 ;       R0 - add1len.rw         Number of digits in first addend string
                                0009   197 ;       R1 - add1addr.ab        Address of first addend string
                                0009   198 ;       R2 - add2len.rw         Number of digits in second addend string
                                0009   199 ;       R3 - add2addr.ab        Address of second addend string
                                0009   200 ;       R4 - sumlen.rw          Number of digits in sum string
                                0009   201 ;       R5 - sumaddr.ab         Address of sum string
                                0009   202 ;
                                0009   203 ; Output Parameters:
                                0009   204 ;
                                0009   205 ;       R0 = 0
                                0009   206 ;       R1 = Address of the byte containing the most significant digit of
                                0009   207 ;            the first addend string
                                0009   208 ;       R2 = 0
                                0009   209 ;       R3 = Address of the byte containing the most significant digit of
                                0009   210 ;            the second addend string
                                0009   211 ;       R4 = 0
                                0009   212 ;       R5 = Address of the byte containing the most significant digit of
                                0009   213 ;            the string containing the sum
                                0009   214 ;
                                0009   215 ; Condition Codes:
                                0009   216 ;
                                0009   217 ;       N <- sum string LSS 0
                                0009   218 ;       Z <- sum string EQL 0
                                0009   219 ;       V <- decimal overflow
                                0009   220 ;       C <- 0
                                0009   221 ;
                                0009   222 ; Register Usage:
                                0009   223 ;
                                0009   224 ;       This routine uses all of the general registers. The condition codes
                                0009   225 ;       are recorded in R11 as the routine executes.
                                0009   226 ;-
                                0009   227
                                0009   228 VAX$ADDP6::
                     0FFF 8F    BB  0009   229          PUSHR   #^M<R0,R1,R2,R3,R4,R5,R6,R7,R8,R9,R10,R11>      ; Save the lot
                          59    D4  000D   230          CLRL    R9                              ; This is addition
                                    000F   231 10$:     ROPRAND_CHECK   R4                      ; Insure that R4 is LEQU 31
                          5B    DC  001A   232          MOVPSL  R11                             ; Get initial PSL
                                    001C   233
                                    001C   234 ; Indicate that the saved R4 must be cleared on the exit path
                                    001C   235
;LJK0045                22 5B    1F  E3  001C    .1          BBCS    #ADD_SUB_V_ZERO_R4,R11,25$      ; Set bit and join common code
;LJK0045                     20    11  0020    .2          BRB     25$                             ; In case we drop through BBCS
```

```
                     0022   239          .SUBTITLE       VAX$SUBP4 - Subtract Packed (4 Operand Format)
                     0022   240 ;+
                     0022   241 ; Functional Description:
                     0022   242 ;
                     0022   243 ;       In 4 operand format, the subtrahend string specified by subtrahend
                     0022   244 ;       length and subtrahend address operands is subtracted from the difference
                     0022   245 ;       string specified by the difference length and difference address
                     0022   246 ;       operands and the difference string is replaced by the result.
                     0022   247 ;
                     0022   248 ; Input Parameters:
                     0022   249 ;
                     0022   250 ;       R0 - sublen.rw          Number of digits in subtrahend string
                     0022   251 ;       R1 - subaddr.ab         Address of subtrahend decimal string
                     0022   252 ;       R2 - diflen.rw          Number of digits in difference string
                     0022   253 ;       R3 - difaddr.ab         Address of difference decimal string
                     0022   254 ;
                     0022   255 ; Output Parameters:
                     0022   256 ;
                     0022   257 ;       R0 = 0
                     0022   258 ;       R1 = Address of the byte containing the most significant digit of
                     0022   259 ;            the subtrahend string
                     0022   260 ;       R2 = 0
                     0022   261 ;       R3 = Address of the byte containing the most significant digit of
                     0022   262 ;            the string containing the difference
                     0022   263 ;
                     0022   264 ; Condition Codes:
                     0022   265 ;
                     0022   266 ;       N <- difference string LSS 0
                     0022   267 ;       Z <- difference string EQL 0
                     0022   268 ;       V <- decimal overflow
                     0022   269 ;       C <- 0
                     0022   270 ;
                     0022   271 ; Register Usage:
                     0022   272 ;
                     0022   273 ;       This routine uses all of the general registers. The condition codes
                     0022   274 ;       are recorded in R11 as the routine executes.
                     0022   275 ;-
                     0022   276
                     0022   277 VAX$SUBP4::
OFFF 8F    BB        0022   278          PUSHR   #^M<R0,R1,R2,R3,R4,R5,R6,R7,R8,R9,R10,R11>      ; Save the lot
59   01    9A        0026   279          MOVZBL  #1,R9                           ; Indicate that this is subtraction
     08    11        0029   280          BRB     20$                             ; Merge with ADDP4 code
```

```
                                002B   282          .SUBTITLE       VAX$ADDP4 - Add Packed (4 Operand Format)
                                002B   283 ;+
                                002B   284 ; Functional Description:
                                002B   285 ;
                                002B   286 ;       In 4 operand format, the addend string specified by  the  addend  length
                                002B   287 ;       and  addend address operands is added to the sum string specified by the
                                002B   288 ;       sum length and sum address operands and the sum string  is  replaced  by
                                002B   289 ;       the result.
                                002B   290 ;
                                002B   291 ; Input Parameters:
                                002B   292 ;
                                002B   293 ;       R0 - addlen.rw          Number of digits in addend string
                                002B   294 ;       R1 - addaddr.ab         Address of addend decimal string
                                002B   295 ;       R2 - sumlen.rw          Number of digits in sum string
                                002B   296 ;       R3 - sumaddr.ab         Address of sum decimal string
                                002B   297 ;
                                002B   298 ; Output Parameters:
                                002B   299 ;
                                002B   300 ;       R0 = 0
                                002B   301 ;       R1 = Address of the byte containing the most significant digit of
                                002B   302 ;            the addend string
                                002B   303 ;       R2 = 0
                                002B   304 ;       R3 = Address of the byte containing the most significant digit of
                                002B   305 ;            the string containing the sum
                                002B   306 ;
                                002B   307 ; Condition Codes:
                                002B   308 ;
                                002B   309 ;       N <- sum string LSS 0
                                002B   310 ;       Z <- sum string EQL 0
                                002B   311 ;       V <- decimal overflow
                                002B   312 ;       C <- 0
                                002B   313 ;
                                002B   314 ; Register Usage:
                                002B   315 ;
                                002B   316 ;       This routine uses all of the general registers. The condition codes
                                002B   317 ;       are recorded in R11 as the routine executes.
                                002B   318 ;-
                                002B   319
;LJK0045                  0264   30   002B    .1 15$:    BSBW    CHECK_WRITE_ACCESS      ; Perform rigorous access check
;LJK0045                    38   11   002E    .2         BRB     30$                     ; String can be written after all
;LJK0045                                0030    .3
                                0030   320 VAX$ADDP4::
                         0FFF  8F   BB 0030   321         PUSHR   #^M<R0,R1,R2,R3,R4,R5,R6,R7,R8,R9,R10,R11>      ; Save the lot
                            59   D4 0034   322         CLRL    R9                      ; This is addition
                                0036   323
                                0036   324 ; The output string, described by R4 and R5, will be the same as the input
                                0036   325 ; string for ADDP4 and SUBP4. It is necessary to explicitly clear R4<31:16>
                                0036   326 ; along this code path so MOVQ R2,R4 will not always work.
                                0036   327
                            54  52   3C 0036   328 20$:     MOVZWL  R2,R4                   ; Set output size equal to input size
                            55  53   D0 0039   329         MOVL    R3,R5                   ; ... and ditto for string addresses
                               5B   DC 003C   330         MOVPSL  R11                     ; Get initial PSL
                                003E   331
                                003E   332 ; Indicate that the saved R4 will be restored on the common exit path
                                003E   333
;LJK0045                  00 5B   1F   E5 003E    .1         BBCC    #ADD_SUB_V_ZERO_R4,R11,25$      ; Clear bit and join common code
```

```
                                0042   336          .SUBTITLE       ADDPx/SUBPx Common Initialization Code
                                0042   337 ;+
                                0042   338 ; All four routines converge at this point and execute common initialization
                                0042   339 ; code until a later decision is made to do addition or subtraction.
                                0042   340 ;
                                0042   341 ;       R4 - Number of digits in destination string
                                0042   342 ;       R5 - Address of destination string
                                0042   343 ;
                                0042   344 ;       R9 - Indicates whether operation is addition or subtraction
                                0042   345 ;               0 => addition
                                0042   346 ;               1 => subtraction
                                0042   347 ;
                                0042   348 ;       R11<31> - Indicates whether this is a 4-operand or 6-operand instruction
                                0042   349 ;               0 => 4-operand (restore saved R4 on exit)
                                0042   350 ;               1 => 6-operand (set R4 to zero on exit)
                                0042   351 ;-
                                0042   352
;LJK0045         04   00   04   F0  0042    .1 25$:     INSV    #PSL$M_Z,#0,#4,R11      ; Set Z-bit, clear the rest in saved PSW
                                5B  0046
-1                              0047   354          ESTABLISH_HANDLER       -       ; Store address of access
                                0047   355                  ARITH_ACCVIO            ;  violation handler
                                004C   356
                                004C   357          ROPRAND_CHECK   R2                      ; Insure that R2 is LEQU 31
                                0054   358          MARK_POINT      ADD_SUB_BSBW_0
                          FFA9' 30  0054   359          BSBW    DECIMAL$STRIP_ZEROS_R2_R3       ; Strip high order zeros from R2/R3
                                0057   360
                                0057   361          ROPRAND_CHECK   R0                      ; Insure that R0 is LEQU 31
                                005F   362          MARK_POINT      ADD_SUB_BSBW_0
                          FF9E' 30  005F   363          BSBW    DECIMAL$STRIP_ZEROS_R0_R1       ; Strip high order zeros from R0/R1
                                0062   364
;LJK0045                        0062    .1 ; Perform the access check on the output string for the worst case, a string
;LJK0045                        0062    .2 ; large enough to accommodate 31 decimal digits. A detailed check using the
;LJK0045                        0062    .3 ; correct byte length of the output string is necessary only if this initial
;LJK0045                        0062    .4 ; probe fails. The more detailed check can be handled out of line.
;LJK0045                        0062    .5
;LJK0045         65   10   00   0D  0062    .6          PROBEW  #0,#16,(R5)             ; Can result string be written?
;LJK0045                     C3   13  0066    .7          BEQL    15$                     ; Branch if no write access allowed
;LJK0045                         0068    .8
                                0068   365 ; Rather than totally confuse the already complicated logic dealing with
                                0068   366 ; different length strings in the add or subtract loop, we will put the
                                0068   367 ; result into an intermediate buffer on the stack. This buffer will be long
                                0068   368 ; enough to handle the worst case so that the addition loop need only concern
                                0068   369 ; itself with the lengths of the two input loops. The required length is 17
                                0068   370 ; bytes, to handle an addition with a carry out of the most significant byte.
                                0068   371 ; We will allocate 20 bytes to maintain whatever alignment the stack has.
                                0068   372
;LJK0045                      7E   7C  0068    .1 30$:     CLRQ    -(SP)                   ; Set aside space for output string
-1                           7E   7C  006A   374          CLRQ    -(SP)                   ; Worst case string needs 16 bytes
                             7E   D4  006C   375          CLRL    -(SP)                   ; Add slack for a CARRY
                   54   04   01   EF  006E   376          EXTZV   #1,#4,R4,R8             ; Get byte count of destination string
                                58  0072
                   7E   55   58   C1  0073   377          ADDL3   R8,R5,-(SP)             ; Save high address end of destination
                     55   18   AE   9E  0077   378          MOVAB   24(SP),R5               ; Point R5 one byte beyond buffer
                                007B   379
                                007B   380 ; The number of minus signs will determine whether the real operation that we
                                007B   381 ; perform is addition or subtraction. That is, two plus signs or two minus
                                007B   382 ; signs will both result in addition, while a plus sign and a minus sign will
```

```
                     007B    383 ; result in subtraction. The addition and subtraction routines have their own
                     007B    384 ; methods for determining the correct sign of the result.
                     007B    385 ;
                     007B    386 ; For the purpose of counting minus signs, we treat subtraction as the
                     007B    387 ; addition of the negative of the input operand. That is, subtraction of a
                     007B    388 ; positive quantity causes the sign to be remembered as minus and counted as
                     007B    389 ; a minus sign while subtraction of a minus quantity stores a plus sign and
                     007B    390 ; counts nothing.
                     007B    391 ;
                     007B    392 ; On input to this code sequence, R9 distinguished addition from subtraction.
                     007B    393 ; On output, it contains either 0, 1, or 2, indicating the total number of
                     007B    394 ; minus signs, real or implied, that we counted.
                     007B    395
     50  04  01  EF 007B    396         EXTZV   #1,#4,R0,R6             ; Get byte count for first input string
                 56  007F
         51  56  C0 0080    397         ADDL    R6,R1                   ; Point R1 to byte containing sign
                     0083    398         MARK_POINT      ADD_SUB_24
       61  F0 8F  8B 0083    399         BICB3   #^B11110000,(R1),R6     ; R6 contains the sign "digit"
                 56  0087
             10 59  E8 0088    400         BLBS    R9,35$                  ; Use second CASE if subtraction
                     008B    401
                     008B    402 ; This case statement is used for addition
                     008B    403
                     008B    404         CASE    R6,TYPE=B,LIMIT=#10,<-  ; Dispatch on sign digit
                     008B    405                 50$,-                   ; 10 => sign is "+"
                     008B    406                 40$,-                   ; 11 => sign is "-"
                     008B    407                 50$,-                   ; 12 => sign is "+"
                     008B    408                 40$,-                   ; 13 => sign is "-"
                     008B    409                 50$,-                   ; 14 => sign is "+"
                     008B    410                 50$,-                   ; 15 => sign is "+"
                     008B    411                 >
                     009B    412
                     009B    413 ; This case statement is used for subtraction
                     009B    414
                     009B    415 35$:    CASE    R6,TYPE=B,LIMIT=#10,<-  ; Dispatch on sign digit
                     009B    416                 40$,-                   ; 10 => treat sign as "-"
                     009B    417                 50$,-                   ; 11 => treat sign as "+"
                     009B    418                 40$,-                   ; 12 => treat sign as "-"
                     009B    419                 50$,-                   ; 13 => treat sign as "+"
                     009B    420                 40$,-                   ; 14 => treat sign as "-"
                     009B    421                 40$,-                   ; 15 => treat sign as "-"
                     009B    422                 >
                     00AB    423
         59  01  D0 00AB    424 40$:    MOVL    #1,R9                   ; Count a minus sign
         56  0D  9A 00AE    425         MOVZBL  #13,R6                  ; The preferred minus sign is 13
             05  11 00B1    426         BRB     60$                     ; Now check second input sign
                     00B3    427
             59  D4 00B3    428 50$:    CLRL    R9                      ; No real minus signs so far
         56  0C  9A 00B5    429         MOVZBL  #12,R6                  ; The preferred minus sign is 12
                     00B8    430
     52  04  01  EF 00B8    431 60$:    EXTZV   #1,#4,R2,R7             ; Get byte count for second input string
                 57  00BC
         53  57  C0 00BD    432         ADDL    R7,R3                   ; Point R3 to byte containing sign
                     00C0    433         MARK_POINT      ADD_SUB_24
       63  F0 8F  8B 00C0    434         BICB3   #^B11110000,(R3),R7     ; R7 contains the sign "digit"
                 57  00C4
                     00C5    435
```

```
                                                B 8
                00C5    436          CASE    R7,TYPE=B,LIMIT=#10,<-  ; Dispatch on sign digit
                00C5    437                  80$,-                   ; 10 => sign is "+"
                00C5    438                  70$,-                   ; 11 => sign is "-"
                00C5    439                  80$,-                   ; 12 => sign is "+"
                00C5    440                  70$,-                   ; 13 => sign is "-"
                00C5    441                  80$,-                   ; 14 => sign is "+"
                00C5    442                  80$,-                   ; 15 => sign is "+"
                00C5    443                  >
                00D5    444
      59  D6    00D5    445 70$:    INCL    R9                      ; Remember that sign was minus
57    0D  9A    00D7    446         MOVZBL  #13,R7                  ; The preferred minus sign is 13
      03  11    00DA    447         BRB     90$                     ; Now check second input sign
                00DC    448
57    0C  9A    00DC    449 80$:    MOVZBL  #12,R7                  ; The preferred minus sign is 12
                00DF    450
  03  59  E9    00DF    451 90$:    BLBC    R9,ADD_PACKED           ; Even parity indicates addition
                00E2    452
 00B3     31    00E2    453         BRW     SUBTRACT_PACKED         ; Odd parity calls for subtraction
                00E5    454
                00E5    455         .DISABLE        LOCAL_BLOCK
```

```
00E5   457          .SUBTITLE       ADD_PACKED - Add Two Packed Decimal Strings
00E5   458 ;+
00E5   459 ; Functional Description:
00E5   460 ;
00E5   461 ;       This routine adds two packed decimal strings whose descriptors
00E5   462 ;       are passed as input parameters and places their sum into another
00E5   463 ;       (perhaps identical) packed decimal string.
00E5   464 ;
00E5   465 ;       At the present time, the result is placed into a 16-byte storage
00E5   466 ;       area while the sum is being evaluated. This drastically reduces
00E5   467 ;       the number of different cases that must be dealt with as each
00E5   468 ;       pair of bytes in the two input strings is added.
00E5   469 ;
00E5   470 ;       The signs of the two input strings have already been dealt with
00E5   471 ;       so this routine performs addition in all cases, even if the original
00E5   472 ;       entry was at SUBP4 or SUBP6. The cases that arrive in this routine
00E5   473 ;       are as follows.
00E5   474 ;
00E5   475 ;                                 R2/R3          R0/R1          result
00E5   476 ;                           +--------------+--------------+--------------+
00E5   477 ;                           |              |              |              |
00E5   478 ;       R2/R3 + R0/R1       |     plus     |     plus     |     plus     |
00E5   479 ;                           |              |              |              |
00E5   480 ;                           +--------------+--------------+--------------+
00E5   481 ;                           |              |              |              |
00E5   482 ;       R2/R3 + R0/R1       |     minus    |     minus    |     minus    |
00E5   483 ;                           |              |              |              |
00E5   484 ;                           +--------------+--------------+--------------+
00E5   485 ;                           |              |              |              |
00E5   486 ;       R2/R3 - R0/R1       |     minus    |     plus     |     minus    |
00E5   487 ;                           |              |              |              |
00E5   488 ;                           +--------------+--------------+--------------+
00E5   489 ;                           |              |              |              |
00E5   490 ;       R2/R3 - R0/R1       |     plus     |     minus    |     plus     |
00E5   491 ;                           |              |              |              |
00E5   492 ;                           +--------------+--------------+--------------+
00E5   493 ;
00E5   494 ;       Note that the correct choice of sign in all four cases is the sign
00E5   495 ;       of the second input string, the one described by R2 and R3.
00E5   496 ;
00E5   497 ; Input Parameters:
00E5   498 ;
00E5   499 ;       R0<4:0> - Number of digits in first input decimal string
00E5   500 ;       R1      - Address of least significant digit of first input
00E5   501 ;                 decimal string (the byte containing the sign)
00E5   502 ;
00E5   503 ;       R2<4:0> - Number of digits in second input decimal string
00E5   504 ;       R3      - Address of least significant digit of second input
00E5   505 ;                 decimal string (the byte containing the sign)
00E5   506 ;
00E5   507 ;       R4<4:0> - Number of digits in output decimal string
00E5   508 ;       R5      - Address of one byte beyond least significant digit of
00E5   509 ;                 intermediate string stored on the stack
00E5   510 ;
00E5   511 ;       R6<3:0> - Sign of first input string in preferred form
00E5   512 ;       R7<3:0> - Sign of second input string in preferred form
00E5   513 ;
```

```
                                00E5   514 ;       R11     - Saved PSL (Z-bit is set, other condition codes are clear)
                                00E5   515 ;
                                00E5   516 ;       (SP)    - Saved R5, address of least significant digit of ultimate
                                00E5   517 ;                 destination string.
                                00E5   518 ;       4(SP)   - Beginning of 20-byte buffer to hold intermediate result
                                00E5   519 ;
                                00E5   520 ; Output Parameters:
                                00E5   521 ;
                                00E5   522 ;       The particular input operation (ADDPx or SUBPx) is completed in
                                00E5   523 ;       this routine. See the routine headers for the four routines that
                                00E5   524 ;       request addition or subtraction for a list of output parameters
                                00E5   525 ;       from this routine.
                                00E5   526 ;-
                                00E5   527
                                00E5   528 ADD_PACKED:
          59    57    90      00E5   529         MOVB    R7,R9                   ; Use sign of second string for output
             03 59    E9      00E8   530         BLBC    R9,10$                  ; Check if sign is negative
          5B    08    88      00EB   531         BISB    #PSL$M_N,R11            ; ... so the saved N-bit can be set
                                00EE   532
                                00EE   533         MARK_POINT      ADD_SUB_24
    56    61    0F    8B      00EE   534 10$:     BICB3   #^B00001111,(R1),R6     ; Get least significant digit to R6
                                00F2   535         MARK_POINT      ADD_SUB_24
    57    63    0F    8B      00F2   536         BICB3   #^B00001111,(R3),R7     ; Get least significant digit to R7
                58    D4      00F6   537         CLRL    R8                      ; Start the add with CARRY off
                   0075   30   00F8   538         BSBW    ADD_PACKED_BYTE_R6_R7   ; Add the two low order digits
                                00FB   539
                                00FB   540 ; The following set of instructions computes the number of bytes in the two
                                00FB   541 ; strings and, if necessary, performs a switch so that R0 and R1 always
                                00FB   542 ; describe the shorter of the two strings.
                                00FB   543
    50    04    01    EF      00FB   544         EXTZV   #1,#4,R0,R0             ; Convert digit count to byte count
                50              00FF
    52    04    01    EF      0100   545         EXTZV   #1,#4,R2,R2             ; Do it for both strings
                52              0104
          52    50    D1      0105   546         CMPL    R0,R2                   ; We want to compare the byte counts
                09    1B      0108   547         BLEQU   20$                     ; Skip the swap if we're already correct
          56    50    7D      010A   548         MOVQ    R0,R6                   ; Save the longer
          50    52    7D      010D   549         MOVQ    R2,R0                   ; Store the shorter on R0 and R1
          52    56    7D      0110   550         MOVQ    R6,R2                   ; ... and store the longer in R2 and R3
          52    5C    C2      0113   551 20$:     SUBL    R0,R2                   ; Make R2 a difference (R2 GEQU 0)
                                0116   552
                                0116   553 ; R0 now contains the number of bytes remaining in the shorter string.
                                0116   554 ; R2 contains the difference in bytes between the two input strings.
                                0116   555
             50    D5      0116   556         TSTL    R0                      ; Does shorter string have any room?
             06    13      0118   557         BEQL    40$                     ; Skip loop if no room at all
                                011A   558
           004D    30      011A   559 30$:     BSBW    ADD_PACKED_BYTE_STRING  ; Add the next two bytes together
         FA 50     F5      011D   560         SOBGTR  R0,30$                  ; Check for end of loop
                                0120   561
             52    D5      0120   562 40$:     TSTL    R2                      ; Does longer string have any room?
             16    13      0122   563         BEQL    70$                     ; Skip next loops if all done
                                0124   564
          0D 58    E9      0124   565 50$:     BLBC    R8,60$                  ; Life is simple if CARRY clear
                                0127   566
             56    D4      0127   567         CLRL    R6                      ; Otherwise, CARRY must propogate
                                0129   568         MARK_POINT      ADD_SUB_24
```

```
                                                E 8
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$:     INSV    R9,#0,#4,@(SP)+         ; The sign can finally be stored
          9E        0150
     5E   14   C0  0151    602          ADDL    #20,SP                  ; Get rid of intermediate buffer
  03 5B   1F   E1  0154    603          BBC     #ADD_SUB_V_ZERO_R4,R11,90$      ; 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
```

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

```
                     016A   672 ;                  R5 - Decreased by one to point to current byte in output string
                     016A   673 ;
                     016A   674 ;                  R8 - Either 0 or 1, reflecting whether this most recent ADD resulted
                     016A   675 ;                       in a CARRY to the next byte.
                     016A   676 ;
                     016A   677 ; Side Effects:
                     016A   678 ;
                     016A   679 ;       R6 and R7 are modified by this routine
                     016A   680 ;
                     016A   681 ;       R0, R2, R4, and R9 (and, of course, R10 and R11) are preserved
                     016A   682 ;       by this routine
                     016A   683 ;
                     016A   684 ; Assumptions:
                     016A   685 ;
                     016A   686 ;       This routine makes two important assumptions.
                     016A   687 ;
                     016A   688 ;       1.  If both of the input bytes contain only legal decimal digits, then
                     016A   689 ;           it is only necessary to subtract 100 at most once to put all
                     016A   690 ;           possible sums in the range 0..99. That is,
                     016A   691 ;
                     016A   692 ;               99 + 99 + 1 = 199 LSS 200
                     016A   693 ;
                     016A   694 ;       2.  The result will be checked in some way to determine whether the
                     016A   695 ;           result is nonzero so that the Z-bit can have its correct setting.
                     016A   696 ;-
                     016A   697
                     016A   698 ADD_PACKED_BYTE_STRING:
                     016A   699
                     016A   700          MARK_POINT      ADD_SUB_BSBW_24
        56   71   9A  016A   701          MOVZBL  -(R1),R6                ; Get byte from first string
                     016D   702          MARK_POINT      ADD_SUB_BSBW_24
        57   73   9A  016D   703          MOVZBL  -(R3),R7                ; Get byte from second string
                     0170   704
                     0170   705 VAX$ADD_PACKED_BYTE_R6_R7::              ; ASHP also uses this routine
                     0170   706 ADD_PACKED_BYTE_R6_R7:
56   0000'CF46   90  0170   707          MOVB    DECIMAL$PACKED_TO_BINARY_TABLE[R6],-
                     0176   708                          R6              ; Convert digits to binary
57   0000'CF47   90  0176   709          MOVB    DECIMAL$PACKED_TO_BINARY_TABLE[R7],-
                     017C   710                          R7              ; Convert digits to binary
        57   56   80  017C   711          ADDB    R6,R7                   ; Form their sum
        57   58   80  017F   712          ADDB    R8,R7                   ; Add CARRY from last step
             58   94  0182   713          CLRB    R8                      ; Assume no CARRY this time
    63 8F    57   91  0184   714          CMPB    R7,#99                  ; Check for CARRY
             07   1B  0188   715          BLEQU   10$                     ; Branch if within bounds
        58   01   90  018A   716          MOVB    #1,R8                   ; Propogate CARRY to next step
    57   64 8F    82  018D   717          SUBB    #100,R7                 ; Put R7 into interval 0..99
75   0000'CF47   90  0191   718 10$:      MOVB    DECIMAL$BINARY_TO_PACKED_TABLE[R7],-
                     0197   719                          -(R5)           ; Store converted sum byte
                   05  0197   720          RSB
```

```
                                H 8
0198    722          .SUBTITLE       SUBTRACT_PACKED - Subtract Two Packed Decimal Strings
0198    723 ;+
0198    724 ; Functional Description:
0198    725 ;
0198    726 ;       This routine takes two packed decimal strings whose descriptors
0198    727 ;       are passed as input parameters, subtracts one string from the
0198    728 ;       other, and places their sum into another (perhaps identical)
0198    729 ;       packed decimal string.
0198    730 ;
0198    731 ;       At the present time, the result is placed into a 16-byte storage
0198    732 ;       area while the difference is being evaluated. This drastically reduces
0198    733 ;       the number of different cases that must be dealt with as each
0198    734 ;       pair of bytes in the two input strings is added.
0198    735 ;
0198    736 ;       The signs of the two input strings have already been dealt with so
0198    737 ;       this routine performs subtraction in all cases, even if the original
0198    738 ;       entry was at ADDP4 or ADDP6.
0198    739 ;
0198    740 ; Input Parameters:
0198    741 ;
0198    742 ;       R0<4:0> - Number of digits in first input decimal string
0198    743 ;       R1      - Address of least significant digit of first input
0198    744 ;                 decimal string (the byte containing the  ign)
0198    745 ;
0198    746 ;       R2<4:0> - Number of digits in second input decimal string
0198    747 ;       R3      - Address of least significant digit of second input
0198    748 ;                 decimal string (the byte containing the sign)
0198    749 ;
0198    750 ;       R4<4:0> - Number of digits in output decimal string
0198    751 ;       R5      - Address of one byte beyond least significant digit of
0198    752 ;                 intermediate string stored on the stack
0198    753 ;
0198    754 ;       R6<3:0> - Sign of first input string in preferred form
0198    755 ;       R7<3:0> - Sign of second input string in preferred form
0198    756 ;
0198    757 ;       R11     - Saved PSL (Z-bit is set, other condition codes are clear)
0198    758 ;
0198    759 ;       (SP)    - Saved R5, address of least significant digit of ultimate
0198    760 ;                 destination string.
0198    761 ;       4(SP)   - Beginning of 20-byte buffer to hold intermediate result
0198    762 ;
0198    763 ; Output Parameters:
0198    764 ;
0198    765 ;       The particular input operation (ADDPx or SUBPx) is completed in
0198    766 ;       this routine. See the routine headers for the four routines that
0198    767 ;       request addition or subtraction for a list of output parameters
0198    768 ;       from this routine.
0198    769 ;
0198    770 ; Algorithm for Choice of Sign:
0198    771 ;
0198    772 ;       The choice of sign for the output string is not nearly so
0198    773 ;       straightforward as it is in the case of addition. One approach that is
0198    774 ;       often taken is to make a reasonable guess at the sign of the result.
0198    775 ;       If the final subtraction causes a BORROW, then the choice was incorrect.
0198    776 ;       The sign must be changed and the result must be replaced by its tens
0198    777 ;       complement.
0198    778 ;
```

```
                                                0198   779 ;       This routine does not guess. Instead, it chooses the input string of
                                                0198   780 ;       the larger absolute magnitude as the minuend for this internal
                                                0198   781 ;       routine and chooses its sign as the sign of the result.
                                                0198   782 ;       This algorithm is actually more efficient than the reasonable
                                                0198   783 ;       guess method and is probably better than a guess method that is never
                                                0198   784 ;       wrong. All complete bytes that are processed in the sign evaluation
                                                0198   785 ;       preprocessing loop are eliminated from consideration in the
                                                0198   786 ;       subtraction loop, which has a higher cost per byte.
                                                0198   787 ;
                                                0198   788 ;       The actual algorithm is as follows. (Note that both input strings have
                                                0198   789 ;       already had leading zeros stripped so their lengths reflect
                                                0198   790 ;       significant digits.)
                                                0198   791 ;
                                                0198   792 ;       1.  If the two strings have unequal lengths, then choose the sign of
                                                0198   793 ;           the string that has the longer length.
                                                0198   794 ;
                                                0198   795 ;       2.  For strings of equal length, choose the sign of the string whose
                                                0198   796 ;           most significant byte is larger in magnitude.
                                                0198   797 ;
                                                0198   798 ;       3.  If the most significant bytes test equal, then decrease the
                                                0198   799 ;           lengths of each string by one byte, drop the previous most
                                                0198   800 ;           significant bytes, and go back to step 2.
                                                0198   801 ;
                                                0198   802 ;       4.  If the two strings test equal, it is not necessary to do any
                                                0198   803 ;           subtraction. The result is identically zero.
                                                0198   804 ;
                                                0198   805 ;       Note that the key to this routine's efficiency is that high order
                                                0198   806 ;       bytes that test equal in this loop are dropped from consideration in
                                                0198   807 ;       the more complicated subtraction loop.
                                                0198   808 ;-
                                                0198   809
                                                0198   810 SUBTRACT_PACKED:
50   04   01   EF                              0198   811         EXTZV   #1,#4,R0,R0             ; Convert digit count to byte count
          50                                    019C
52   04   01   EF                              019D   812         EXTZV   #1,#4,R2,R2             ; Do it for both strings
          52                                    01A1
     52   50   D1                              01A2   813         CMPL    R0,R2                   ; We want to compare the byte counts
          3C   1F                              01A5   814         BLSSU   40$                     ; R0/R1 represent the smaller string
          2A   1A                              01A7   815         BGTRU   30$                     ; R2/R3 represent the smaller string
                                                01A9   816
                                                01A9   817   The two input strings have an equal number of bytes. Compare magnitudes to
                                                01A9   818 , determine which string is really larger. If the two strings test equal, then
                                                01A9   819 ; skip the entire subtraction loop.
                                                01A9   820
58   51   50   C3                              01A9   821         SUBL3   R0,R1,R8                ; Point R8 to low address end of R0/R1
59   53   52   C3                              01AD   822         SUBL3   R2,R3,R9                ; Point R9 to low address end of R2/R3
          50   D5                              01B1   823         TSTL    R0                      ; See if both strings have zero bytes
          0C   13                              01B3   824         BEQL    20$                     ; Still need to check low order digit
                                                01B5   825
                                                01B5   826         MARK_POINT      ADD_SUB_24
     89   88   91                              01B5   827 10$:     CMPB    (R8)+,(R9)+             ; Compare most significant bytes
          29   1F                              01B8   828         BLSSU   40$                     ; R0/R1 represent the smaller string
          17   1A                              01BA   829         BGTRU   30$                     ; R2/R3 represent the smaller string
          52   D7                              01BC   830         DECL    R2                      ; Keep R2 in step with R0
        F4 50   F5                              01BE   831         SOBGTR  R0,10$                  ; ... which gets decremented here
                                                01C1   832
                                                01C1   833 ; At this point, we have reduced both input strings to single bytes that
```

```
                     01C1    834 ; contain a sign "digit" and may contain a digit in the high order nibble
                     01C1    835 ; if the original digit counts were nonzero.
                     01C1    836
                     01C1    837             MARK_POINT      ADD_SUB_24
58   68   0F   8B    01C1    838 20$:        BICB3   #^B00001111,(R8),R8     ; Look only at digit, ignoring sign
                     01C5    839             MARK_POINT      ADD_SUB_24
59   69   0F   8B    01C5    840             BICB3   #^B00001111,(R9),R9     ; Get the digit from the other string
     59   58   91    01C9    841             CMPB    R8,R9                   ; Compare these digits
          15   1F    01CC    842             BLSSU   40$                     ; R0/R1 represent the smaller string
          03   1A    01CE    843             BGTRU   30$                     ; R2/R3 represent the smaller string
                     01D0    844
                     01D0    845 ; The two strings have identical magnitudes. Enter the end processing code
                     01D0    846 ; with the intermediate result unchanged (that is, zero).
                     01D0    847
               FF6A  31    01D0    848             BRW     ADD_SUBTRACT_EXIT       ; Join the common completion code
                     01D3    849
                     01D3    850 ; The string described by R0 and R1 has the larger magnitude. Choose its sign.
                     01D3    851 ; Then swap the two string descriptors so that the main subtraction loops
                     01D3    852 ; always have R2 and R3 describing the larger string. Note that the use of
                     01D3    853 ; R6 and R7 as scratch leaves R7<31:8> in an UNPREDICTABLE state.
                     01D3    854
59   56   90    01D3    855 30$:        MOVB    R6,R9                   ; Load preferred sign into R9
56   50   7D    01D6    856             MOVQ    R0,R6                   ; Save the longer
50   52   7D    01D9    857             MOVQ    R2,R0                   ; Store the shorter on R0 and R1
52   56   7D    01DC    858             MOVQ    R6,R2                   ; ... and store the longer in R2 and R3
          57   D4    01DF    859             CLRL    R7                      ; Insure that R7<31:8> is zero
          03   11    01E1    860             BRB     50$                     ; Continue along common code path
                     01E3    861
                     01E3    862 ; The string described by P2 and R3 has the larger magnitude. Choose its sign.
                     01E3    863
59   57   90    01E3    864 40$:        MOVB    R7,R9                   ; Load preferred sign into R9
                     01E6    865
52   50   C2    01E6    866 50$:        SUBL    R0,R2                   ; Make R2 a difference (R2 GEQU 0)
     03   59    E9    01E9    867             BLBC    R9,60$                  ; Check if sign is negative
5B   08   88    01EC    868             BISB    #PSL$M_N,R11            ; ... so the saved N-bit can be set
                     01EF    869
                     01EF    870             MARK_POINT      ADD_SUB_24
56   61   0F   88    01EF    871 60$:        BICB3   #^B00001111,(R1),R6     ; Get least significant digit to R6
                     01F3    872             MARK_POINT      ADD_SUB_24
57   63   0F   88    01F3    873             BICB3   #^B00001111,(R3),R7     ; Get least significant digit to R7
          58   D4    01F7    874             CLRL    R8                      ; Start subtracting with BORROW off
     0032   30    01F9    875             BSBW    SUB_PACKED_BYTE_R6_R7   ; Subtract the two low order digits
                     01FC    876
                     01FC    877 ; R0 contains the number of bytes remaining in the smaller string
                     01FC    878 ; R2 contains the difference in bytes between the two input strings
                     01FC    879
          50   D5    01FC    880             TSTL    R0                      ; Does smaller string have any room?
          06   13    01FE    881             BEQL    80$                     ; Skip loop if no room at all
                     0200    882
     0025   30    0200    883 70$:        BSBW    SUB_PACKED_BYTE_STRING  ; Subtract the next two bytes
   FA 50   F5    0203    884             SOBGTR  R0,70$                  ; Check for end of loop
                     0206    885
          52   D5    0206    886 80$:        TSTL    R2                      ; Does one of the strings have more?
          16   13    0208    887             BEQL    110$                    ; Skip next loops if all done
                     020A    888
0D   58   E9    020A    889 90$:        BLBC    R8,100$                 ; Life is simple if BORROW clear
                     020D    890
```

```
                     56   D4  020D    891          CLRL    R6                              ; Otherwise, BORROW must propogate
                                 020F    892          MARK_POINT      ADD_SUB_24
                 57  73   9A  020F    893          MOVZBL  -(R3),R7                        ; So subtract BORROW from single string
                   0019   30  0212    894          BSBW    SUB_PACKED_GETS_R6_R7           ; Use the special entry point
                 F2 52    F5  0215    895          SOBGTR  R2,90$                          ; Check for this string exhausted
                    06    11  0218    896          BRB     110$                            ; Join common completion code
                                 021A    897
                                 021A    898          MARK_POINT      ADD_SUB_24
                 75  73   90  021A    899 100$:    MOVB    -(R3),-(R5)                     ; Simply move src to dst if no BORROW
                   FA 52   F5  021D    900          SOBGTR  R2,100$                         ; ... until we're all done
                               0220    901
                               0220    902 110$:
                               0220    903
                               0220    904 ;;; ********** BEGIN TEMP **********
                               0220    905 ;;;
                               0220    906 ;;; THE FOLLOWING HALT INSTRUCTION SHOULD BE REPLACED WITH THE CORRECT
                               0220    907 ;;; ABORT CODE.
                               0220    908 ;;;
                               0220    909 ;;; THE HALT IS SIMILAR TO THE
                               0220    910 ;;;
                               0220    911 ;;;     MICROCODE CANNOT GET HERE
                               0220    912 ;;;
                               0220    913 ;;; ERRORS THAT OTHER IMPLEMENTATIONS USE.
                               0220    914 ;;;
                     58   D5  0220    915          tstl    r8                              ; If BORROW is set here, we blew it
                     01   13  0222    916          beql    120$                            ; Branch out if OK
                          00  0224    917          halt                                    ; This will cause an OPCDEC exception
                               0225    918 120$:
                               0225    919 ;;;
                               0225    920 ;;; ********** END TEMP **********
                               0225    921
                   FF15   31  0225    922          BRW     ADD_SUBTRACT_EXIT               ; Join common completion code
```

```
0228   924          .SUBTITLE       SUB_PACKED_BYTE - Subtract Two Bytes Containing Decimal Digi
0228   925 ;+
0228   926 ; Functional Description:
0228   927 ;
0228   928 ;       This routine takes as input two bytes containing decimal digits and
0228   929 ;       produces a byte containing their difference. This result is stored in
0228   930 ;       the output string. Each of the input bytes is converted to a binary
0228   931 ;       number (with a table-driven conversion), the first number is
0228   932 ;       subtracted from the second, and the difference is converted back to
0228   933 ;       two decimal digits stored in a byte.
0228   934 ;
0228   935 ;       This routine makes no provisions for bytes that contain illegal
0228   936 ;       decimal digits. We are using the UNPREDICTABLE statement in the
0228   937 ;       architectural description of the decimal instructions to its fullest.
0228   938 ;
0228   939 ;       The bytes that contain a pair of packed decimal digits can either
0228   940 ;       exist in packed decimal strings located by R1 and R3 or they can
0228   941 ;       be stored directly in registers. In the former case, the digits must
0228   942 ;       be extracted from registers before they can be used in later operations
0228   943 ;       because the difference will be used as an index register.
0228   944 ;
0228   945 ; For entry at SUB_PACKED_BYTE_STRING:
0228   946 ;
0228   947 ;       Input Parameters:
0228   948 ;
0228   949 ;               R1  - Address one byte beyond byte containing subtrahend
0228   950 ;               R3  - Address one byte beyond byte containing minuend
0228   951 ;               R5  - Address one byte beyond location to store difference
0228   952 ;
0228   953 ;               R8  - BORROW from previous byte (R8 is either 0 or 1)
0228   954 ;
0228   955 ;       Implicit Input:
0228   956 ;
0228   957 ;               R6 - Scratch
0228   958 ;               R7 - Scratch
0228   959 ;
0228   960 ;       Output Parameters:
0228   961 ;
0228   962 ;               R1 - Decreased by one to point to current byte
0228   963 ;                    in subtrahend string
0228   964 ;               R3 - Decreased by one to point to current byte
0228   965 ;                    in minuend string
0228   966 ;               R5 - Decreased by one to point to current byte
0228   967 ;                    in difference string
0228   968 ;
0228   969 ;               R8 - Either 0 or 1, reflecting whether this most recent
0228   970 ;                    subtraction resulted in a BORROW from the next byte.
0228   971 ;
0228   972 ; For entry at SUB_PACKED_BYTE_R6_R7:
0228   973 ;
0228   974 ;       Input Parameters:
0228   975 ;
0228   976 ;               R6<7:0>  - Byte containing decimal digit pair for subtrahend
0228   977 ;               R6<31:8> - MBZ
0228   978 ;               R7<7:0>  - Byte containing decimal digit pair for minuend
0228   979 ;               R7<31:8> - MBZ
0228   980 ;
```

```
                                                M 8
0228   981 ;                       R5 - Address one byte beyond location to store difference
0228   982 ;
0228   983 ;                       R8 - BORROW from subtraction of previous byte
0228   984 ;                            (R8 is either 0 or 1)
0228   985 ;
0228   986 ;       Output Parameters:
0228   987 ;
0228   988 ;                       R5 - Decreased by one to point to current byte
0228   989 ;                            in difference string
0228   990 ;
0228   991 ;                       R8 - Either 0 or 1, reflecting whether this most recent
0228   992 ;                            subtraction resulted in a BORROW from the next byte.
0228   993 ;
0228   994 ; Side Effects:
0228   995 ;
0228   996 ;       R6 and R7 are modified by this routine
0228   997 ;
0228   998 ;       R0, R2, R4, and R9 (and, of course, R10 and R11) are preserved
0228   999 ;       by this routine
0228  1000 ;
0228  1001 ; Assumptions:
0228  1002 ;
0228  1003 ;       This routine makes two important assumptions.
0228  1004 ;
0228  1005 ;       1.  If both of the input bytes contain only legal decimal digits, then
0228  1006 ;           it is only necessary to add 100 at most once to put all
0228  1007 ;           possible differences in the range 0..99. That is,
0228  1008 ;
0228  1009 ;                   0 - 99 - 1 = -100
0228  1010 ;
0228  1011 ;       2.  The result will be checked in some way to determine whether the
0228  1012 ;           result is nonzero so that the Z-bit can have its correct setting.
0228  1013 ;-
0228  1014
0228  1015 SUB_PACKED_BYTE_STRING:
0228  1016
0228  1017          MARK_POINT      ADD_SUB_BSBW_24
      56   71   9A 0228  1018          MOVZBL  -(R1),R6                ; Get byte from first string
022B  1019          MARK_POINT      ADD_SUB_BSBW_24
      57   73   9A 022B  1020          MOVZBL  -(R3),R7                ; Get byte from second string
022E  1021
022E  1022 SUB_PACKED_BYTE_R6_R7:
56  0000'CF46   90 022E  1023          MOVB    DECIMAL$PACKED_TO_BINARY_TABLE[R6],-
0234  1024                                      R6                      ; Convert digits to binary
57  0000'CF47   90 0234  1025          MOVB    DECIMAL$PACKED_TO_BINARY_TABLE[R7],-
023A  1026                                      R7                      ; Convert digits to binary
      57   56   82 023A  1027          SUBB    R6,R7                   ; Form their difference
      57   58   82 023D  1028          SUBB    R8,R7                   ; Include BORROW from last step
           04   19 0240  1029          BLSS    10$                     ; Branch if need to BORROW
           58   94 0242  1030          CLRB    R8                      ; No BORROW next time
           07   11 0244  1031          BRB     20$                     ; Join common exit code
0246  1032
   57   64 8F   80 0246  1033 10$:     ADDB    #100,R7                 ; Put R7 into interval 0..99
      58   01   90 024A  1034          MOVB    #1,R8                   ; Propogate BORROW to next step
024D  1035
75  0000'CF47   90 024D  1036 20$:     MOVB    DECIMAL$BINARY_TO_PACKED_TABLE[R7],-
0253  1037                                      -(R5)                   ; Store converted sum byte
```

```
05  0253  1038          RSB
```

```
                     0254  1040          .SUBTITLE       STORE_RESULT - Store Decimal String
                     0254  1041 ;+
                     0254  1042 ; Functional Description:
                     0254  1043 ;
                     0254  1044 ;       This routine takes a packed decimal string that typically contains
                     0254  1045 ;       the result of an arithmetic operation and stores it in another
                     0254  1046 ;       decimal string whose descriptor is specified as an input parameter
                     0254  1047 ;       to the original arithmetic operation.
                     0254  1048 ;
                     0254  1049 ;       The string is stored from the high address end (least significant
                     0254  1050 ;       digits) to the low address end (most significant digits). This order
                     0254  1051 ;       allows all of the special cases to be handled in the simplest fashion.
                     0254  1052 ;
                     0254  1053 ; Input Parameters:
                     0254  1054 ;
                     0254  1055 ;       R1      - Address one byte beyond high address end of input string
                     0254  1056 ;                 (Note that this string must be at least 17 bytes long.)
                     0254  1057 ;
                     0254  1058 ;       R4<4:0> - Number of digits in ultimate destination
                     0254  1059 ;       R5      - Address one byte beyond destination string
                     0254  1060 ;
                     0254  1061 ;       R11     - Contains saved condition codes
                     0254  1062 ;
                     0254  1063 ; Implicit Input:
                     0254  1064 ;
                     0254  1065 ;       The input string must be at lerst 17 bytes long to contain a potential
                     0254  1066 ;       carry out of the highest digit when doing an add of two large numbers.
                     0254  1067 ;       This carry out of the last byte will be detected and reported as a
                     0254  1068 ;       decimal overflow, either as an exception or simply by setting the V-bit.
                     0254  1069 ;
                     0254  1070 ;       The least significant digit (highest addressed byte) cannot contain a
                     0254  1071 ;       sign digit because that would cause the Z-bit to be incorrectly cleared.
                     0254  1072 ;
                     0254  1073 ; Output Parameters:
                     0254  1074 ;
                     0254  1075 ;       R11<PSL$V_Z> - Cleared if a nonzero digit is stored in output string
                     0254  1076 ;       R11<PSL$V_V> - Set if a nonzero digit is detected after the output
                     0254  1077 ;                      string is exhausted
                     0254  1078 ;
                     0254  1079 ;       A portion of the result (dictated by the size of R4 on input) is
                     0254  1080 ;       moved to the destination string.
                     0254  1081 ;-
                     0254  1082
                     0254  1083 STORE_RESULT:
          54    D6   0254  1084          INCL    R4                              ; Want number of "complete" bytes in
54   FF 8F    78   0256  1085          ASHL    #-1,R4,R0                       ;  output string
          50        025A
          0B    13   025B  1086          BEQL    30$                             ; Skip first loop if none
                     025D  1087
                     025D  1088          MARK_POINT      ADD_SUB_BSBW_24
    75    71    90   025D  1089 10$:     MOVB    -(R1),-(R5)                     ; Move the next complete byte
          03    13   0260  1090          BEQL    20$                             ; Check whether to clear Z-bit
 5B       04    8A   0262  1091          BICB    #PSL$M_Z,R11                    ; Clear Z-bit if nonzero
   F5 50       F5   0265  1092 20$:     SOBGTR  R0,10$                          ; Keep going?
                     0268  1093
     10 54     E9   0268  1094 30$:     BLBC    R4,50$                          ; Was original R4 odd? Branch if yes
                     026B  1095          MARK_POINT      ADD_SUB_BSBW_24
```

```
                71   F0 8F    8B  026B  1096          BICB3   #^B11110000,-(R1),-(R5) ; If R4 was even, store half a byte
                                75  026F
                     03    13   0270  1097          BEQL    40$                     ; Need to check for zero here, too
                5B   04    8A   0272  1098          BICB    #PSL$M_Z,R11            ; Clear Z-bit if nonzero
                                0275  1099          MARK_POINT      ADD_SUB_BSBW_24
                61   F0 8F    93   0275  1100 40$:    BITB    #^B11110000,(R1)        ; If high order nibble is nonzero,
                     13    12   0279  1101          BNEQ    70$                     ; ... then overflow has occurred
                                027B  1102
                                027B  1103 ; The entire destination has been stored. We must now check whether any of
                                027B  1104 ; the remaining input string is nonzero and set the V-bit if nonzero is
                                027B  1105 ; detected. Note that at least one byte of the output string has been examined
                                027B  1106 ; in all cases already. This makes the next byte count calculation correct.
                                027B  1107
                     54    D7   027B  1108 50$:    DECL    R4                      ; Restore R4 to its original self
           54   04    01    EF   027D  1109          EXTZV   #1,#4,R4,R0             ; Extract a byte count
                     50         0281
           50   10    50    83   0282  1110          SUBB3   R0,#16,R0               ; Loop count is 16 minus byte count
                                0286  1111
                                0286  1112 ; Note that the loop count can never be zero because we are testing a 17-byte
                                0286  1113 ; string and the largest output string can be 16 bytes long.
                                0286  1114
                                0286  1115          MARK_POINT      ADD_SUB_BSBW_24
                     71    95   0286  1116 60$:    TSTB    -(R1)                   ; Check next byte for nonzero
                     04    12   0288  1117          BNEQ    70$                     ; Nonzero means overflow has occurred
                   F9 50   F5   028A  1118          SOBGTR  R0,60$                  ; Check for end of this loop
                                028D  1119
                           05   028D  1120          RSB                             ; This is return path for no overflow
                                028E  1121
                5B   02    88   028E  1122 70$:    BISB    #PSL$M_V,R11            ; Indicate that overflow has occurred
                           05   0291  1123          RSB                             ; ... and return to the caller
```

```
:LJK0045                0292     .1          .SUBTITLE - CHECK_WRITE_ACCESS - Check Writability of Decimal String
:LJK0045                0292     .2 ;+
:LJK0045                0292     .3 ; Functional Description:
:LJK0045                0292     .4 ;
:LJK0045                0292     .5 ;       The ADDP4 and SUBP4 instructions are unique in that they are the only
:LJK0045                0292     .6 ;       two instructions that read and write the same packed decimal string.
:LJK0045                0292     .7 ;       They are, in fact, implemented as ADDP6 and SUBP6 where the second
:LJK0045                0292     .8 ;       input string, ADDend2 or MINuend, and the result string, SUM or
:LJK0045                0292     .9 ;       DIFference, are the same. But ADDP6 and SUBP6, as well as all other
:LJK0045                0292    .10 ;       packed decimal instructions except ADDP4 and SUBP4, produce
:LJK0045                0292    .11 ;       UNPREDICTABLE results when an output string overlaps any input string.
:LJK0045                0292    .12 ;       With this interpretation, ADDP4 and SUBP4 are the only packed decimal
:LJK0045                0292    .13 ;       instructions that permit overlapping packed decimal strings. (Note
:LJK0045                0292    .14 ;       that the result string, SUM or DIFference, may not overlap the first
:LJK0045                0292    .15 ;       input string, ADDend or SUBtrahend.)
:LJK0045                0292    .16 ;
:LJK0045                0292    .17 ;       The implementation of ADDP4 and SUBP4, interpreted as ADDP6 and SUBP6
:LJK0045                0292    .18 ;       with overlapping strings, needs to protect itself from modifying
:LJK0045                0292    .19 ;       memory until the entire instruction can execute to completion.
:LJK0045                0292    .20 ;       Otherwise, the instruction will be restarted with a different initial
:LJK0045                0292    .21 ;       state than it first had. This is accomplished by probing the result
:LJK0045                0292    .22 ;       string for write access before execution begins. This routine receives
:LJK0045                0292    .23 ;       control if that PROBEW fails.
:LJK0045                0292    .24 ;
:LJK0045                0292    .25 ;       In the interest of simplicity, the PROBEW that is executed always uses
:LJK0045                0292    .26 ;       16 as the byte count of the result string. This routine can then be
:LJK0045                0292    .27 ;       called not only when the output string is inaccessible but also when
:LJK0045                0292    .28 ;
:LJK0045                0292    .29 ;              The output string is shorter than 16 bytes.
:LJK0045                0292    .30 ;
:LJK0045                0292    .31 ;              The address of the output string is within 16 bytes of the
:LJK0045                0292    .32 ;              end of the page.
:LJK0045                0292    .33 ;
:LJK0045                0292    .34 ;              The page containing the string is writable.
:LJK0045                0292    .35 ;
:LJK0045                0292    .36 ;              The next page is not writable.
:LJK0045                0292    .37 ;
:LJK0045                0292    .38 ;       This routine distinguishes inaccessible strings from this rare case of
:LJK0045                0292    .39 ;       a PROBEW failure.
:LJK0045                0292    .40 ;
:LJK0045                0292    .41 ;       In other words, this routine checks the write accessibility of packed
:LJK0045                0292    .42 ;       decimal strings. If the entire string is writable, control is passed
:LJK0045                0292    .43 ;       back to the caller where emulation continues. If the string is not
:LJK0045                0292    .44 ;       writable, an access violation is generated.
:LJK0045                0292    .45 ;
:LJK0045                0292    .46 ; Input Parameters:
:LJK0045                0292    .47 ;
:LJK0045                0292    .48 ;       R4 - Digit count of result string
:LJK0045                0292    .49 ;       R5 - Address of most significant digit in result string
:LJK0045                0292    .50 ;
:LJK0045                0292    .51 ; Output Parameters:
:LJK0045                0292    .52 ;
:LJK0045                0292    .53 ;       None
:LJK0045                0292    .54 ;
:LJK0045                0292    .55 ; Implicit Ouptut:
:LJK0045                0292    .56 ;
:LJK0045                0292    .57 ;       If the string with its correct byte count is writable, control
```

```
:LJK0045                                0292   .58 ;       is returned to the caller.
:LJK0045                                0292   .59 ;
:LJK0045                                0292   .60 ;       If the string with its correct digit count is not writable, then the
:LJK0045                                0292   .61 ;       string is accessed to force an access violation to occur. This will
:LJK0045                                0292   .62 ;       cause control to be transferred eventually to the access violation
:LJK0045                                0292   .63 ;       handler at the end of this module that will back out the instruction
:LJK0045                                0292   .64 ;       so that it can be restarted.
:LJK0045                                0292   .65 ;
:LJK0045                                0292   .66 ; Assumption:
:LJK0045                                0292   .67 ;
:LJK0045                                0292   .68 ;       This routine assumes that the check for illegal digit count has
:LJK0045                                0292   .69 ;       already been made so that R4 is between 0 and 31 inclusive.
:LJK0045                                0292   .70 ;
:LJK0045                                0292   .71 ; No:e:
:LJK0045                                0292   .72 ;
:LJK0045                                0292   .73 ;       It is necessary to actually touch the inaccessible page so that the
:LJK0045                                0292   .74 ;       correct reason mask is generated. Logic that uses the PROBEW
:LJK0045                                0292   .75 ;       instruction can determine the inaccessible virtual address but is
:LJK0045                                0292   .76 ;       unable to distinguish, for example, between length violations and page
:LJK0045                                0292   .77 ;       protection violations.
:LJK0045                                0292   .78 ;-
:LJK0045                                0292   .79
:LJK0045                                0292   .80 CHECK_WRITE_ACCESS:
:LJK0045                    54   DD     0292   .81         PUSHL   R4                      ; Save the digit count
:LJK0045          54   04   01   EF     0294   .82         EXTZV   #1,#4,R4,R4             ; Convert digits to bytes
                            54          0298
:LJK0045                    54   D6     0299   .83         INCL    R4                      ; Count the byte that contains the sign
:LJK0045          65   54   00   0D     029B   .84         PROBEW  #0,R4,(R5)              ; Cneck writability of the string
:LJK0045                    04   13     029F   .85         BEQL    10$                     ; Branch if string cannot be written
:LJK0045               54   8E   D0     02A1   .86         MOVL    (SP)+,R4                ; Restore saved R4
:LJK0045                    05          02A4   .87         RSB                             ; Return to caller. String is OK
:LJK0045                                02A5   .88
:LJK0045                                02A5   .89 ; The first and last bytes of the string are touched (written) with an
:LJK0045                                02A5   .90 ;
:LJK0045                                02A5   .91 ;       ADDB2   #0,xxx
:LJK0045                                02A5   .92 ;
:LJK0045                                02A5   .93 ; instruction. This instruction causes the correct access violation reason
:LJK0045                                02A5   .94 ; mask to be generated but does not modify the contents of locations that are
:LJK0045                                02A5   .95 ; accessible. Note that at least one of the following two ADDB2 instructions
:LJK0045                                02A5   .96 ; is guaranteed to generate an access violation, transferring control in a
:LJK0045                                02A5   .97 ; rather complicated way to the ADD_SUB_BSBW_4 access violation handler.
:LJK0045                                02A5   .98
:LJK0045                                02A5   .99         MARK_POINT      ADD_SUB_BSBW_4
:LJK0045            65   00   80     02A5  .100 10$:     ADDB2   #0,(R5)                 ; Touch the first byte
:LJK0045                                02A8  .101         MARK_POINT      ADD_SUB_BSBW_4
:LJK0045          6544   00   80     02A8  .102         ADDB2   #0,(R5)[R4]             ; Touch the last byte
:LJK0045                                02AC  .103
:LJK0045                                02AC  .104 ; We should never reach here unless the PROBE instruction is broken. We
:LJK0045                                02AC  .105 ; will leave this code path in for now but remove it at the same time we
:LJK0045                                02AC  .106 ; change the other two HALT instructions in this module into software
:LJK0045                                02AC  .107 ; generated machine checks exceptions.
:LJK0045                                02AC  .108
:LJK0045                                02AC  .109 ;;; ********** BEGIN TEMP **********
:LJK0045                                02AC  .110 ;;;
:LJK0045                                02AC  .111 ;;; THE FOLLOWING HALT INSTRUCTION SHOULD BE REPLACED WITH THE CORRECT
:LJK0045                                02AC  .112 ;;; ABORT CODE.
:LJK0045                                02AC  .113 ;;;
```

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

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

```
                                02AD  1182 ;       output string has been completely filled are the indication of decimal
                                02AD  1183 ;       overflow.
                                02AD  1184 ;
                                02AD  1185 ;       The purpose of this method of storage is to avoid decimal/binary or
                                02AD  1186 ;       even byte/longword conversions during the calculation of intermediate
                                02AD  1187 ;       results.
                                02AD  1188 ;
                                02AD  1189 ;    2. Trailing zeros are removed from the larger string. All zeros in
                                02AD  1190 ;       the shorter string are eliminated in the sense that no arithmetic
                                02AD  1191 ;       is performed. The output array pointer is simply advanced to point
                                02AD  1192 ;       to the next higher array element.
                                02AD  1193 ;-
                                02AD  1194
                                02AD  1195 VAX$MULP::
                    0FFF 8F     BB  02AD  1196         PUSHR   #^M<R0,R1,R2,R3,R4,R5,R6,R7,R8,R9,R10,R11>      ; Save the lot
                                02B1  1197
                                02B1  1198         ESTABLISH_HANDLER       -       ; Store address of access
                                02B1  1199                 ARITH_ACCVIO            ;  violation handler
                                02B6  1200
                                02B6  1201         ROPRAND_CHECK   R4                      ; Insure that R4 is LEQU 31
                                02C1  1202
                                02C1  1203         ROPRAND_CHECK   R2                      ; Insure that R2 is LEQU 31
                                02C9  1204         MARK_POINT      MULP_BSBW_0
                        FD34'   30  02C9  1205         BSBW    DECIMAL$STRIP_ZEROS_R2_R3       ; Strip high order zeros from R2/R3
                                02CC  1206
                                02CC  1207         ROPRAND_CHECK   R0                      ; Insure that R0 is LEQU 31
                                02D4  1208         MARK_POINT      MULP_BSBW_0
                        FD29'   30  02D4  1209         BSBW    DECIMAL$STRIP_ZEROS_R0_R1       ; Strip high order zeros from R0/R1
                                02D7  1210
                50 04   01      EF  02D7  1211         EXTZV   #1,#4,R0,R0             ; Convert digit count to byte count
                                50      02DB
                        50      D6  02DC  1212         INCL    R0                      ; Include least significant digit
                                02DE  1213
                52 04   01      EF  02DE  1214         EXTZV   #1,#4,R2,R2             ; Convert digit count to byte count
                                52      02E2
                        52      D6  02E3  1215         INCL    R2                      ; Include least significant digit
                                02E5  1216
                    52  50      D1  02E5  1217         CMPL    R0,R2                   ; See which string is larger
                        08      1A  02E8  1218         BGTRU   3$                      ; R2/R3 describes the longer string
                    58  52      7D  02EA  1219         MOVQ    R2,R8                   ; R8 and R9 describe the longer string
                    7E  50      7D  02ED  1220         MOVQ    R0,-(SP)                ; Shorter string descriptor also saved
                        06      11  02F0  1221         BRB     6$
                                02F2  1222
                    58  50      7D  02F2  1223 3$:     MOVQ    R0,R8                   ; R8 and R9 describe the longer string
                    7E  52      7D  02F5  1224         MOVQ    R2,-(SP)                ; Shorter string descriptor also saved
                                02F8  1225
                                02F8  1226 ; Create space for the output array on the stack (32 longwords of zeros)
                                02F8  1227
                    50  08      D0  02F8  1228 6$:     MOVL    #8,R0                   ; Eight pairs of quadwords
                                02FB  1229
                        7E      7C  02FB  1230 10$:    CLRQ    -(SP)                   ; Clear one pair
                        7E      7C  02FD  1231         CLRQ    -(SP)                   ; ... and another
                  F9    50      F5  02FF  1232         SOBGTR  R0,10$                  ; Do all eight pairs
                                0302  1233
                    57  5E      D0  0302  1234         MOVL    SP,R7                   ; Store beginning of output array in R7
                                0305  1235
                                0305  1236 ; The longer input array will be stored on the stack as an array of
```

```
                                                                I 9
                     0305  1237 ; longwords. Each array element contains a number between 0 and 99,
                     0305  1238 ; representing a pair of digits in the original packed decimal string.
                     0305  1239 ; Because the units digit is stored with the sign in packed decimal format,
                     0305  1240 ; it is necessary to shift the number as we store it. This is accomplished by
                     0305  1241 ; multiplying the number by ten.
                     0305  1242 ;
                     0305  1243 ; The longer array is described by R8 (byte count) and R9 (address of most
                     0305  1244 ; significant digit pair).
                     0305  1245
          55   58   59   C1  0305  1246          ADDL3   R9,R8,R5                ; Point R5 beyond sign digit
               54   58   D0  0309  1247          MOVL    R8,R4                   ; R4 contains the loop count
                     030C  1248
                     030C  1249 ; An array of longwords is allocated on the stack. R3 starts out pointing
                     030C  1250 ; at the longword beyond the top of the stack. The first remainder, guaranteed
                     030C  1251 ; to be zero, is "stored" here. The rest of the digit pairs are stored safely
                     030C  1252 ; below the top of the stack.
                     030C  1253
               53   58   CE  030C  1254          MNEGL   R8,R3                   ; Stack grows toward lower addresses
            5E    6E43   DE  030F  1255          MOVAL   (SP)[R3],SP             ; Allocate the space
          53   5E   04   C3  0313  1256          SUBL3   #4,SP,R3                ; Point R3 at next lower longword
                     0317  1257
                     0317  1258          MARK_POINT      MULP_R8
               51   75   9A  0317  1259 20$:     MOVZBL  -(R5),R1                ; Get next digit pair
        51   0000'CF41   9A  031A  1260          MOVZBL  DECIMAL$PACKED_TO_BINARY_TABLE[R1],-
                     0320  1261                          R1                      ; Convert digits to binary
          52   51   0A   7A  0320  1262          EMUL    #10,R1,R2,R0            ; Multiply by 10
                    50        0324
           00000064   8F   7B  0325  1263          EDIV    #100,R0,R2,(R3)+        ; Divide by 100
          83   52   50        032B
                 E6   54   F5  032E  1264          SOBGTR  R4,20$
                     0331  1265
               63   52   D0  0331  1266          MOVL    R2,(R3)                 ; Store final quotient
               59   5E   D0  0334  1267          MOVL    SP,R9                   ; Remember array address in R9
                  6E48   DF  0337  1268          PUSHAL  (SP)[R8]                ; Store start of fixed size area
                     033A  1269
                     033A  1270 ; Check for trailing zeros in the input array stored on the stack. If any are
                     033A  1271 ; present, they are removed and the product array is adjusted accordingly.
                     033A  1272
                     033A  1273 30$:     TSTL    (R9)+                   ; Is next number zero?
                     08   12  033C  1274          BNEQ    40$                     ; Leave loop if nonzero
               57    04   C0  033E  1275          ADDL    #4,R7                   ; Advance output pointer to next element
                 F6   58   F5  0341  1276          SOBGTR  R8,30$                  ; Keep going
                     0344  1277
                     0344  1278 ; If we drop through the loop, then the entire input array is zero. There is
                     0344  1279 ; no need to perform any arithmetic because the product will be zero (and the
                     0344  1280 ; output array on the stack starts out as zero). The only remaining work is
                     0344  1281 ; to store the result in the output string and set the condition codes.
                     0344  1282
                     0344  1283          BRB     70$                     ; Exit to end processing
                     0346  1284
                     0346  1285 ; Now multiply the input array by each successive digit pair. In order to
                     0346  1286 ; allow R10 to continue to locate ARITH_ACCVIO while we execute this loop, it
                     0346  1287 ; is necessary to perform a small amount of register juggling. In essence,
                     0346  1288 ; R8 and R9 switch the identity of the string that they describe.
                     0346  1289
               59   04   C2  0346  1290 40$:     SUBL    #4,R9                   ; Readjust input array pointer
               7E   58   7D  0349  1291          MOVQ    R8,-(SP)                ; Save R8/R9 descriptor on stack
```

```
                58    08 AE    D0  034C  1292          MOVL    8(SP),R8                        ; Point R8 at start of 32-longword array
             58   0080 C8    7D  0350  1293          MOVQ    <32*4>(R8),R8                   ; Get descriptor that follows that array
                  59   58    C0  0355  1294          ADDL2   R8,R9                           ; Point R9 beyond sign byte
                                0358  1295
                  53   87    DE  0358  1296 50$:     MOVAL   (R7)+,R3                        ; Output array address to R3
                                035B  1297          MARK_POINT      MULP_AT_SP
                  51   79    9A  035B  1298          MOVZBL  -(R9),R1                        ; Next digit pair to R1
           56   0000'CF41    9A  035E  1299          MOVZBL  DECIMAL$PACKED_TO_BINARY_TABLE[R1],-
                                0364  1300                          R6                      ; Convert digits to binary
                    06   13    0364  1301          BEQL    60$                             ; Skip the work if zero
                  54   6E    7D  0366  1302          MOVQ    (SP),R4                         ; Input array descriptor to R4/R5
                     0104   30  0369  1303          BSBW    EXTEND_STRING_MULTIPLY          ; Do the work
                    E9 58    F5  036C  1304 60$:     SOBGTR  R8,50$                          ; Any more multiplier digits?
                                036F  1305
                  5E   08    C0  036F  1306          ADDL    #8,SP                           ; Discard saved long string descriptor
                                0372  1307
                  5E   6E    D0  0372  1308 70$:     MOVL    (SP),SP                         ; Remove input array from stack
                                0375  1309
                                0375  1310 ; At this point, the product string is located in a 32-longword array on
                                0375  1311 ; the top of the stack. Each longword corresponds to a pair of digits in
                                0375  1312 ; the output string. As digits are removed from the stack, they are checked
                                0375  1313 ; for nonzero to obtain the correct setting of the Z-bit. After the output
                                0375  1314 ; string has been filled, the remainder of the product string is removed from
                                0375  1315 ; the stack. If a nonzero result is detected at this stage, the V-bit is set.
                                0375  1316
                  59   20    D0  0375  1317          MOVL    #32,R9                          ; Set up array counter
                54   0098 CE    7D  0378  1318          MOVQ    < <32*4> + -                    ; Skip over 32-longword array
                                037D  1319                    <2*4>  + -                    ;  and saved string descriptor
                                037D  1320                    <4*4> >(SP),R4                ;  to retrieve original R4 and R5
```

```
                                037D  1322          .SUBTITLE       Common Exit Path for VAX$MULP and VAX$DIVP
                                037D  1323 ;+
                                037D  1324 ; The code for VAX$MULP and VAX$DIVP merges at this point. The result is stored
                                037D  1325 ; in an array of longwords at the top of the stack. The size of this array is
                                037D  1326 ; stored in R9. The original R4 and R5 heve been retrieved from the stack.
                                037D  1327 ;
                                037D  1328 ; Input Parameters:
                                037D  1329 ;
                                037D  1330 ;       R4 - Contains byte count of destination string in R4 <1:4>
                                037D  1331 ;       R5 - Address of most significant digit of destination string
                                037D  1332 ;       R9 - Count of longwords in result array on stack
                                037D  1333 ;
                                037D  1334 ;       Contents of result array
                                037D  1335 ;
                                037D  1336 ; Implicit Input:
                                037D  1337 ;
                                037D  1338 ;       Signs of two input factors (multiplier and multiplicand or
                                037D  1339 ;           divisor and dividend)
                                037D  1340 ;-
                                037D  1341
                                037D  1342 MULTIPLY_DIVIDE_EXIT:
                          5B    DC  037D  1343          MOVPSL  R11                     ; Get current PSL
              04    00    04    F0  037F  1344          INSV    #PSL$M_Z,#0,#4,R11      ; Clear all codes except Z-bit
                          5B        0383
                                0384  1345          ESTABLISH_HANDLER       -       ; Store address of access
                                0384  1346                  ARITH_ACCVIO            ;  violation handler again
              54    04    01    EF  0389  1347          EXTZV   #1,#4,R4,R3             ; Excess byte count to R3
                          53        038D
                          38    13  038E  1348          BEQL    125$                    ; Skip to single digit code
              57    55    53    C1  0390  1349          ADDL3   R3,R5,R7                ; Remember address of sign byte
              55    57    01    C1  0394  1350          ADDL3   #1,R7,R5                ; Point R5 beyond end of product string
                                0398  1351
                    51    8E    D0  0398  1352 80$:     MOVL    (SP)+,R1                ; Remove next value from stack
                          03    13  039B  1353          BEQL    90$                     ; Do not clear Z-bit if zero
                    5B    04    8A  039D  1354          BICB2   #PSL$M_Z,R11            ; Clear Z-bit
                                03A0  1355
                                03A0  1356          MARK_POINT      MULP_DIVP_R9
           75 0000'CF41         90  03A0  1357 90$:     MOVB    DECIMAL$BINARY_TO_PACKED_TABLE[R1],-
                                03A6  1358                          -(R5)               ; Store converted sum byte
                          59    D7  03A6  1359          DECL    R9                      ; One less element on the stack
                          1C    15  03A8  1360          BLEQ    116$                    ; Exit loop if result array exhausted
                       EB 53    F5  03AA  1361          SOBGTR  R3,80$                  ; Keep going?
                                03AD  1362
                    22 54       E9  03AD  1363 100$:    BLBC    R4,120$                 ; Different for even digit count
                                03B0  1364
                                03B0  1365 ; The output string consists of an odd number of digits. A complete digit
                                03B0  1366 ; pair can be stored in the most significant (lowest addressed) byte of
                                03B0  1367 ; the product string.
                                03B0  1368
                    51    8E    D0  03B0  1369          MOVL    (SP)+,R1                ; Remove next value from stack
                          03    13  03B3  1370          BEQL    110$                    ; Do not clear Z-bit if zero
                    5B    04    8A  03B5  1371          BICB2   #PSL$M_Z,R11            ; Clear Z-bit
                                03B8  1372
                                03B8  1373          MARK_POINT      MULP_DIVP_R9
           75 0000'CF41         90  03B8  1374 110$.    MOVB    DECIMAL$BINARY_TO_PACKED_TABLE[R1],-
                                03BE  1375                          -(R5)               ; Store converted sum byte
                          59    D7  03BE  1376          DECL    R9                      ; One less element on the stack
```

```
                04   15  03C0  1377          BLEQ    116$                    ; Exit loop if result array exhausted
                38   11  03C2  1378          BRB     140$                    ; Perform overflow check
                         03C4  1379
                         03C4  1380 ; This loop executes if the result array has fewer elements than the output
                         03C4  1381 ; string. The remaining bytes in the output string are filled with zeros.
                         03C4  1382 ; There is no need for an overflow check.
                         03C4  1383
                         03C4  1384          MARK_POINT      MULP_DIVP_8
                75   94  03C4  1385 114$:    CLRB    -(R5)                   ; Store another zero byte
              FB 53  F4  03C6  1386 116$:    SOBGEQ  R3,114$                 ; Any more room in output string
                         03C9  1387
                38   11  03C9  1388          BRB     150$                    ; Determine sign of result
                         03CB  1389
                         03CB  1390 ; This code path is used in the case where the output digit count is 0 or 1.
                         03CB  1391 ; R5 must be advanced
                         03CB  1392
             57  55   D0  03CB  1393 125$:    MOVL    R5,R7                   ; Remember address of output sign byte
                 55   D6  03CE  1394          INCL    R5                      ; Advance R5 so common code can be used
                 DB   11  03D0  1395          BRB     100$                    ; Join common code path
                         03D2  1396
                         03D2  1397 ; The output string consists of an even number of digits. Only the low order
                         03D2  1398 ; nibble is stored in the most significant (lowest addresses) byte. A zero is
                         03D2  1399 ; stored in the high order nibble. If the high order digit would have been
                         03D2  1400 ; nonzero, the V-bit is set and the overflow check is bypassed because there
                         03D2  1401 ; are faster ways to clean the stack if we do not have to check for nonzero
                         03D2  1402 ; at the same time.
                         03D2  1403
               51   8E   D0  03D2  1404 120$:    MOVL    (SP)+,R1                ; Remove next value from stack
          51   0000'CF41   90  03D5  1405          MOVB    DECIMAL$BINARY_TO_PACKED_TABLE[R1],-
                         03D8  1406                          R1                  ; Obtain converted sum byte
                         03D8  1407          MARK_POINT      MULP_DIVP_R9
            51   F0 8F    88  03DB  1408          BICB3   #^XF0,R1,-(R5)          ; Store byte, clearing high order nibble
                 75           03DF
                 03   13  03E0  1409          BEQL    130$                    ; Do not clear Z-bit if zero
              5B 04   8A  03E2  1410          BICB2   #PSL$M_Z,R11            ; Clear Z-bit
          51   F0 8F   93  03E5  1411 130$:    BITB    #^XF0,R1                ; Is high order nibble nonzero?
                 06   12  03E9  1412          BNEQ    133$                    ; Yes, go set overflow bit
                 59   D7  03EB  1413          DECL    R9                      ; One less element on the stack
                 D7   15  03ED  1414          BLEQ    116$                    ; Exit loop if result array exhausted
                 08   11  03EF  1415          BRB     140$                    ; Check rest of result array for nonzero
                         03F1  1416
                         03F1  1417 ; If we detect overflow, we need to adjust R9 to reflect the nonzero longword
                         03F1  1418 ; removed from the stack before we enter the next code block that sets the
                         03F1  1419 ; V-bit and cleans off the stack based on the contents of R9.
                         03F1  1420
                 59   D7  03F1  1421 133$:    DECL    R9                      ; One more longword removed from stack
                         03F3  1422
                         03F3  1423 ; A nonzero digit has been discovered in a position that cannot be stored in
                         03F3  1424 ; the output string. Set the V-bit, remove the rest of the product array from
                         03F3  1425 ; the stack, and join the exit processing in the code that determines the sign
                         03F3  1426 ; of the product.
                         03F3  1427
              5B 02   88  03F3  1428 135$:    BISB    #PSL$M_V,R11            ; Set the overflow bit
          5E   6E49   DE  03F6  1429          MOVAL   (SP)[R9],SP             ; Clean off remaining product string
                 07   11  03FA  1430          BRB     150$                    ; Go to code that determines the sign
                         03FC  1431
                         03FC  1432 ; The remainder of the product array must be removed from the stack. A nonzero
```

```
                                03FC  1433 ; result causes the V-bit to be set and the rest of the loop to be skipped.
                                03FC  1434 ; Note that there is always a nonzero loop count remaining at this point.
                                03FC  1435
                     8E   D5    03FC  1436 140$:    TSTL    (SP)+                   ; Is next longword zero?
                     F1   12    03FE  1437          BNEQ    133$                    ; No, leave loop
                  F9 59   F5    0400  1438          SOBGTR  R9,140$
                                0403  1439
                                0403  1440 ; The final product string has been stored and the V- and Z-bits have their
                                0403  1441 ; correct settings. The sign of the product must be determined from the
                                0403  1442 ; signs of the two input strings. Opposite signs produce a negative product.
                                0403  1443 ; Same signs (in any representation) produce a plus sign in the output string.
                                0403  1444
                     5E   08    C0    0403  1445 150$:    ADDL    #8,SP                   ; Discard saved string descriptor
                     56   0C    D0    0406  1446          MOVL    #12,R6                  ; Assume final result is positive
                     50   6E    7D    0409  1447          MOVQ    (SP),R0                 ; Retrieve original R0/R1 pair
                50   04   01    EF    040C  1448          EXTZV   #1,#4,R0,R0             ; Get byte count for first input string
                          50          0410
                     51   50    C0    0411  1449          ADDL    R0,R1                   ; Point R1 to byte containing sign
                                0414  1450          MARK_POINT      MULP_DIVP_0
                  61   F0 8F    88    0414  1451          BICB3   #^B11110000,(R1),R0     ; R0 contains the sign "digit"
                          50          0418
                                0419  1452
                                0419  1453          CASE    R0,TYPE=B,LIMIT=#10,<-  ; Dispatch on sign digit
                                0419  1454                  220$,-                  ; 10 => sign is "+"
                                0419  1455                  210$,-                  ; 11 => sign is "-"
                                0419  1456                  220$,-                  ; 12 => sign is "+"
                                0419  1457                  210$,-                  ; 13 => sign is "-"
                                0419  1458                  220$,-                  ; 14 => sign is "+"
                                0419  1459                  220$,-                  ; 15 => sign is "+"
                                0419  1460                  >
                                0429  1461
                     54   01    D0    0429  1462 210$:    MOVL    #1,R4                   ; Count a minus sign
                          02    11    042C  1463          BRB     230$                    ; Now check second input sign
                                042E  1464
                     54   D4    042E  1465 220$:    CLRL    R4                      ; No real minus signs so far
                                0430  1466
                52   08 AE    7D    0430  1467 230$:    MOVQ    8(SP),R2                ; Retrieve original R2/R3 pair
             52 04   01    EF    0434  1468          EXTZV   #1,#4,R2,R2             ; Get byte count for second input string
                          52          0438
                     53   52    C0    0439  1469          ADDL    R2,R3                   ; Point R3 to byte containing sign
                                043C  1470          MARK_POINT      MULP_DIVP_0
                  63   F0 8F    88    043C  1471          BICB3   #^B11110000,(R3),R2     ; R2 contains the sign "digit"
                          52          0440
                                0441  1472
                                0441  1473          CASE    R2,TYPE=B,LIMIT=#10,<-  ; Dispatch on sign digit
                                0441  1474                  250$,-                  ; 10 => sign is "+"
                                0441  1475                  240$,-                  ; 11 => sign is "-"
                                0441  1476                  250$,-                  ; 12 => sign is "+"
                                0441  1477                  240$,-                  ; 13 => sign is "-"
                                0441  1478                  250$,-                  ; 14 => sign is "+"
                                0441  1479                  250$,-                  ; 15 => sign is "+"
                                0441  1480                  >
                                0451  1481
                     54   D6    0451  1482 240$:    INCL    R4                      ; Remember that sign was minus
                  09 54   E9    0453  1483 250$:    BLBC    R4,260$                 ; Even parity indicates positive result
               10 5B   02    E0    0456  1484          BBS     #PSL$V_Z,R11,270$       ; Step out of line for minus zero check
                  5B   08    88    045A  1485          BISB    #PSL$M_N,R11            ; Set N-bit in saved PSW
```

```
                    56   D6  045D  1486 255$:   INCL    R6                      ; Change sign to minus
                                045F  1487
                                045F  1488         MARK_POINT      MULP_DIVP_0
          04    00  56   F0  045F  1489 260$:   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  1500          .SUBTITLE       EXTEND_STRING_MULTIPLY - Multiply a String by a Number
0470  1501 ;+
0470  1502 ; Functional Description:
0470  1503 ;
0470  1504 ;       This routine multiplies an array of numbers (each array element LEQU
0470  1505 ;       99) by a number (also LEQU 99). The resulting product array is added
0470  1506 ;       to another array, each of whose elements is also LEQU 99.
0470  1507 ;
0470  1508 ; Input Parameters:
0470  1509 ;
0470  1510 ;       R3 - Pointer to output array
0470  1511 ;       R4   Input array size
0470  1512 ;       R5 - Input array address
0470  1513 ;       R6 - Multiplier
0470  1514 ;
0470  1515 ; Output Parameters:
0470  1516 ;
0470  1517 ;       None
0470  1518 ;
0470  1519 ; Implicit Output:
0470  1520 ;
0470  1521 ;       The output array is altered.
0470  1522 ;
0470  1523 ;       An intermediate product array is produced by multiplying each input
0470  1524 ;       array element by the multiplier. Each product array element is then
0470  1525 ;       added to the corresponding output array element.
0470  1526 ;
0470  1527 ; Side Effects:
0470  1528 ;
0470  1529 ;       R3, R4, and R5 are modified by this routine.
0470  1530 ;
0470  1531 ;       R6 is preserved.
0470  1532 ;
0470  1533 ;       R0, R1, and R2 are used as scratch registers. R0 and R1 contain the
0470  1534 ;       quadword result of EMUL that is then passed into EDIV.
0470  1535 ;
0470  1536 ; Assumptions:
0470  1537 ;
0470  1538 ;       This routine assumes that all array elements lie in the range from 0
0470  1539 ;       to 99 inclusive. (This is true if all input strings contain only legal
0470  1540 ;       decimal digits.) The arithmetic performed by this routine will
0470  1541 ;       maintain this assumption. That is,
0470  1542 ;
0470  1543 ;                                 input array element    LEQU 99
0470  1544 ;                       times                multiplier    LEQU 99
0470  1545 ;                               ------------------------
0470  1546 ;                                          product                LEQU 99*99
0470  1547 ;                       plus                 carry    LEQU 99
0470  1548 ;                               ------------------------
0470  1549 ;                                   modified product           LEQU 99*100
0470  1550 ;                       plus    old output array element    LEQU 99
0470  1551 ;                               ------------------------
0470  1552 ;                               new output array element           LEQU 99*101 = 9999
0470  1553 ;
0470  1554 ;       A number LEQU 9999, when divided by 100, is guaranteed to produce both
0470  1555 ;       a quotient and a remainder LEQU 99.
0470  1556 ;-
```

```
                                0470  1557
                                0470  1558 EXTEND_STRING_MULTIPLY:
                        52  D4  0470  1559          CLRL    R2                      ; Initial carry is zero
                                0472  1560
                    52  85  56  7A  0472  1561 10$:    EMUL    R6,(R5)+,R2,R0          ; Form modified product (R0 LEQU 9900)
                            50      0476
                        50  63  C0  0477  1562          ADDL2   (R3),R0                 ; Add old output array element
                  00000064  8F  7B  047A  1563          EDIV    #100,R0,R2,(R3)+        ; Remainder to output array
                    83  52  50      0480
                                0483  1564                                          ; Quotient becomes carry
                          EC 54  F5  0483  1565          SOBGTR  R4,10$                  ; Keep going?
                                0486  1566
                                0486  1567 ; This remaining code looks more complicated than it actually is. In the
                                0486  1568 ; usual case, the routine exits immediately. In the event that a carry
                                0486  1569 ; occurs, one additional entry in the output array will be modified. Only in
                                0486  1570 ; the rare case of an output array consisting of a string of 99s will any
                                0486  1571 ; significant looping occur.
                                0486  1572
                            63  52  C0  0486  1573          ADDL2   R2,(R3)                 ; Add final carry
                                63  D1  0489  1574 20$:    CMPL    (R3),#100               ; Do we overflow into next digit pair?
                  00000064  8F      048B
                                01  1E  0490  1575          BGEQU   30$                     ; Branch if carry
                                05  0492  1576          RSB                             ; Otherwise, all done
                                0493  1577
                  00000064  8F  C2  0493  1578 30$:    SUBL    #100,(R3)+              ; Readjust entry and advance pointer
                                83      0499
                                63  D6  049A  1579          INCL    (R3)                    ; Propogate carry
                                EB  11  049C  1580          BRB     20$                     ; ... and test this entry for overflow
```

```
049E  1582          .SUBTITLE       VAX$DIVP - Divide Packed
049E  1583 ;+
049E  1584 ; Functional Description:
049E  1585 ;
049E  1586 ;       The dividend string specified by the dividend length and dividend
049E  1587 ;       address operands is divided by the divisor string specified by the
049E  1588 ;       divisor length and divisor address operands.  The quotient string
049E  1589 ;       specified by the quotient length and quotient address operands is
049E  1590 ;       replaced by the result.
049E  1591 ;
049E  1592 ; Input Parameters:
049E  1593 ;
049E  1594 ;       R0 - divrlen.rw         Number of digits in divisor string
049E  1595 ;       R1 - divraddr.ab        Address of divisor string
049E  1596 ;       R2 - divdlen.rw         Number of digits in dividend string
049E  1597 ;       R3 - divdaddr.ab        Address of dividend string
049E  1598 ;       R4 - quolen.rw          Number of digits in quotient string
049E  1599 ;       R5 - quoaddr.ab         Address of quotient string
049E  1600 ;
049E  1601 ; Output Parameters:
049E  1602 ;
049E  1603 ;       R0 = 0
049E  1604 ;       R1 = Address of the byte containing the most significant digit of
049E  1605 ;            the divisor string
049E  1606 ;       R2 = 0
049E  1607 ;       R3 = Address of the byte containing the most significant digit of
049E  1608 ;            the dividend string
049E  1609 ;       R4 = 0
049E  1610 ;       R5 = Address of the byte containing the most significant digit of
049E  1611 ;            the string containing the quotient
049E  1612 ;
049E  1613 ; Condition Codes:
049E  1614 ;
049E  1615 ;       N <- quotient string LSS 0
049E  1616 ;       Z <- quotient string EQL 0
049E  1617 ;       V <- decimal overflow
049E  1618 ;       C <- 0
049E  1619 ;
049E  1620 ; Register Usage:
049E  1621 ;
049E  1622 ;       This routine uses all of the general registers. The condition codes
049E  1623 ;       are computed at the end of the instruction as the final result is
049E  1624 ;       stored in the quotient string. R11 is used to record the condition
049E  1625 ;       codes.
049E  1626 ;
049E  1627 ; Algorithm:
049E  1628 ;
049E  1629 ;       This algorithm is the straightforward approach described in
049E  1630 ;
049E  1631 ;               The Art of Computer Programming
049E  1632 ;               Second Edition
049E  1633 ;
049E  1634 ;               Volume 2 / Seminumerical Algorithms
049E  1635 ;               Donald E. Knuth
049E  1636 ;
049E  1637 ;               1981
049E  1638 ;               Addison-Wesley Publishing Company
```

```
                                                ;                Reading, Massachusetts
                                                ;
                                                ; Notes:
                                                ;
                                                ;       The choice of a longword array to store the quotient deserves a
                                                ;       comment. In VAX$MULP, a longword array was used because its elements
                                                ;       were used directly by MULP and DIVP instructions. The use of longwords
                                                ;       eliminated the need to convert back and forth between longwords and
                                                ;       bytes. In this routine, the QUOTIENT_DIGIT routine returns its result
                                                ;       in a register, which result can easily be stored in whatever way is
                                                ;       convenient. By using longwords instead of bytes, this routine can use
                                                ;       the same end processing code as MULP, a sizeable savings in code.
                                                ;-

                                                        .ENABLE         LOCAL_BLOCK

                                                ;+
                                                ; This code path is entered if the divisor is zero.
                                                ;
                                                ; Input Parameter:
                                                ;
                                                ;       (SP) - Return PC
                                                ;
                                                ; Output Parameters:
                                                ;
                                                ;       0(SP) - SRM$K_FLT_DIV_T (Arithmetic trap code)
                                                ;       4(SP) - Final state PSL
                                                ;       8(SP) - Return PC
                                                ;
                                                ; Implicit Output:
                                                ;
                                                ;       Control passes through this code to VAX$REFLECT_TRAP.
                                                ;-

                                                DIVIDE_BY_ZERO:
OFFF 8F     BA                                          POPR    #^M<R0,R1,R2,R3,R4,R5,R6,R7,R8,R9,R10,R11>
                                                                                        ; Restore registers and reset SP
     7E     DC                                          MOVPSL  -(SP)                   ; Save final PSL on stack
     04     DD                                          PUSHL   #SRM$K_FLT_DIV_T        ; Store arithmetic trap code
   FB57'    31                                          BRW     VAX$REFLECT_TRAP        ; Report exception

                                                ; If the divisor contains more nonzero digits than the dividend, then the
                                                ; quotient will be identically zero. Set up the stack and the registers (R4,
                                                ; R5, and R9)  so that the exit code will be entered to produce this result.

     7E     D4                                  1$:     CLRL    -(SP)                   ; Fake a quotient digit
59   01     D0                                          MOVL    #1,R9                   ; Count that digit
   FECC     31                                          BRW     MULTIPLY_DIVIDE_EXIT    ; Store the zero in the output string

                                                VAX$DIVP::
OFFF 8F     BB                                          PUSHR   #^M<R0,R1,R2,R3,R4,R5,R6,R7,R8,R9,R10,R11>      ; Save the lot

                                                        ESTABLISH_HANDLER       -       ; Store address of access
                                                                ARITH_ACCVIO            ;  violation handler

                                                        ROPRAND_CHECK   R4                      ; Insure that R4 is LEQU 31
```

```
                                04C5  1696          ROPRAND_CHECK   R2                      ; Insure that R2 is LEQU 31
                                04CD  1697          MARK_POINT      DIVP_BSBW_0
                     FB30'  30  04CD  1698          BSBW    DECIMAL$STRIP_ZEROS_R2_R3       ; Strip high order zeros from R2/R3
                                04D0  1699
                                04D0  1700          ROPRAND_CHECK   R0                      ; Insure that R0 is LEQU 31
                                04D8  1701          MARK_POINT      DIVP_BSBW_0
                     FB25'  30  04D8  1702          BSBW    DECIMAL$STRIP_ZEROS_R0_R1       ; Strip high order zeros from R0/R1
                                04DB  1703
                                04DB  1704 ; Insure that the divisor is not zero. Because leading zeros have already
                                04DB  1705 ; been eliminated, the divisor can only be zero if R0 is 0 (zero length
                                04DB  1706 ; strings are identically zero) or 1 (R1 contains a sign digit in the low
                                04DB  1707 ; order nibble and zero in the high order nibble). Note that an exception
                                04DB  1708 ; will not be generated if an even length string has an illegal nonzero digit
                                04DB  1709 ; stored in its most significant nibble (including an illegal form of a zero
                                04DB  1710 ; length string.
                                04DB  1711
          50    04    01    EF  04DB  1712          EXTZV   #1,#4,R0,R0             ; Convert divisor digit count to bytes
                          50    04DF
                          06    12  04E0  1713          BNEQ    10$                     ; Skip zero divisor check unless zero
                                04E2  1714          MARK_POINT      DIVP_0
            61    F0 8F     93  04E2  1715          BITB    #^B11110000,(R1)        ; Check for zero in ones digit
                          B6    13  04E6  1716          BEQL    DIVIDE_BY_ZERO          ; Generate exception if zero
                                04E8  1717
                                04E8  1718 ; This routine chooses to do its work with a fair amount of internal storage,
                                04E8  1719 ; all of it allocated on the stack. The quotient is stored as it is computed,
                                04E8  1720 ; in a 16-longword array. The dividend and divisor are stored as longword arrays,
                                04E8  1721 ; with each array element storing a digit pair from the original packed
                                04E8  1722 ; decimal string. The numerator digits are shifted by one digit (multiplied
                                04E8  1723 ; by ten) so that the quotient has its digits correctly placed, leaving room
                                04E8  1724 ; for a sign in the low order nibble of the least significant byte. A scratch
                                04E8  1725 ; array is also allocated on the stack to accommodate intermediate results
                                04E8  1726 ; of the QUOTIENT_DIGIT routine.
                                04E8  1727
                          50    D6  04E8  1728 10$:    INCL    R0                      ; Include least significant digit
                    58    50    7D  04EA  1729          MOVQ    R0,R8                   ; Let R8 and R9 describe the divisor
                                04ED  1730
          52    04    01    EF  04ED  1731          EXTZV   #1,#4,R2,R2             ; Convert dividend digit count to bytes
                          52    04F1
                          52    D6  04F2  1732          INCL    R2                      ; Include least significant digit
                    7E    52    7D  04F4  1733          MOVQ    R2,-(SP)                ; Save dividend descriptor on stack
                                04F7  1734
          56    52    50    C3  04F7  1735          SUBL3   R0,R2,R6                ; Calculate main loop count
                          AC    1F  04FB  1736          BLSSU   1$                      ; Quotient will be zero
                          56    D6  04FD  1737          INCL    R6                      ; One extra digit is always there
                                04FF  1738
                                04FF  1739 ; Allocate R6 longwords of zero on the stack
                                04FF  1740
                    50    56    D0  04FF  1741          MOVL    R6,R0                   ; Let R0 be the loop counter
                          7E    D4  0502  1742 15$:    CLRL    -(SP)                   ; Set aside another quotient digit
                       FB 50    F5  0504  1743          SOBGTR  R0,15$                  ; Keep going
                                0507  1744
                    57    5E    D0  0507  1745          MOVL    SP,R7                   ; Remember where this array starts
                                050A  1746
                                050A  1747 ; The divisor will be stored on the stack as an array of
                                050A  1748 ; longwords. Each array element contains a number between 0 and 99,
                                050A  1749 ; representing a pair of digits in the original packed decimal string.
                                050A  1750 ; Because the units digit is stored with the sign in packed decimal format,
```

```
                                050A  1751 ; it is necessary to shift the number as we store it. This is accomplished by
                                050A  1752 ; multiplying the number by ten.
                                050A  1753 ;
                                050A  1754 ; The divisor string is described by R8 (byte count) and R9 (address of most
                                050A  1755 ; significant digit pair).
                                050A  1756
          55    58    59    C1  050A  1757          ADDL3   R9,R8,R5                ; Point R5 beyond sign digit
                54    58    D0  050E  1758          MOVL    R8,R4                   ; R4 contains the loop count
                                0511  1759
                                0511  1760 ; Put in an extra digit place for the divisor. This allows several common
                                0511  1761 ; subroutines to be used when operating on the divisor string.
                                0511  1762
                      7E    D4  0511  1763          CLRL    -(SP)                   ; Set aside a place holder
                                0513  1764
                                0513  1765 ; An array of longwords is allocated on the stack. R3 starts out pointing
                                0513  1766 ; at the longword beyond the top of the stack. The first remainder, guaranteed
                                0513  1767 ; to be zero, is "stored" here. The rest of the digit pairs are stored safely
                                0513  1768 ; below the top of the stack.
                                0513  1769
             53    58    CE  0513  1770          MNEGL   R8,R3                   ; Stack grows toward lower addresses
          5E    6E43    DE  0516  1771          MOVAL   (SP)[R3],SP             ; Allocate the space
       53    5E    04    C3  051A  1772          SUBL3   #4,SP,R3                ; Point R3 at next lower longword
                                051E  1773
                                051E  1774          MARK_POINT      DIVP_R6_R7
             51    75    9A  051E  1775 20$:     MOVZBL  -(R5),R1                ; Get next digit pair
    51   0000'CF41    9A  0521  1776          MOVZBL  DECIMAL$PACKED_TO_BINARY_TABLE[R1],-
                                0527  1777                  R1                  ; Convert digits to binary
       52    51    0A    7A  0527  1778          EMUL    #10,R1,R2,R0            ; Multiply by 10
                   50        052B
     00000064    8F    7B  052C  1779          EDIV    #100,R0,R2,(R3)+        ; Divide by 100
    83    52    50          0532
             E6    54    F5  0535  1780          SOBGTR  R4,20$
                               0538  1781
                               0538  1782 ; There are two cases where the final quotient (contents of R2) is zero.
                               0538  1783 ; In these cases, the number of nonzero digit pairs in the divisor array is
                               0538  1784 ; smaller by one than the number of bytes containing the original packed decimal
                               0538  1785 ; string. One case is a divisor string with an even number of digits. The
                               0538  1786 ; second case is a divisor string with an odd number of digits but the most
                               0538  1787 ; significant digit is zero (essentially a variant of the first case). The
                               0538  1788 ; simplest way to handle all of these cases is to decrement R8, the divisor
                               0538  1789 ; counter, if R2 is zero. Note that previous checks for a zero divisor
                               0538  1790 ; prevent R8 from going to zero.
                               0538  1791
          63    52    D0  0538  1792          MOVL    R2,(R3)                 ; Store final quotient
                0A    12  053B  1793          BNEQ    25$                     ; Leave well enough alone if nonzero
                56    D6  053D  1794          INCL    R6                      ; One more quotient digit
          57    04    C2  053F  1795          SUBL    #4,R7                   ; Make room for it
                58    D7  0542  1796          DECL    R8                      ; Count one less divisor "digit"
                01    12  0544  1797          BNEQ    25$
                               0546  1798
                               0546  1799 ;;; ********** BEGIN TEMP **********
                               0546  1800 ;;;
                               0546  1801 ;;; THE FOLLOWING HALT INSTRUCTION SHOULD BE REPLACED WITH THE CORRECT
                               0546  1802 ;;; ABORT CODE.
                               0546  1803 ;;;
                               0546  1804 ;;; THE HALT IS SIMILAR TO THE
                               0546  1805 ;;;
```

```
                        0546  1806 ;;;     MICROCODE CANNOT GET HERE
                        0546  1807 ;;;
                        0546  1808 ;;; ERRORS THAT OTHER IMPLEMENTATIONS USE.
                        0546  1809 ;;;
                     00 0546  1810         halt                            ; This will cause an OPCDEC exception
                        0547  1811 ;;;
                        0547  1812 ;;; ********** END TEMP **********
                        0547  1813
                  59 5E D0 0547  1814 25$:    MOVL    SP,R9                   ; R9 locates low order divisor digit
                        054A  1815
                        054A  1816 ; The dividend is stored on the stack as an array of longwords. It does not
                        054A  1817 ; have its digit pairs shifted so that this storage loop is simpler. An extra
                        054A  1818 ; place is set aside in the event that it is necessary to normalize the
                        054A  1819 ; dividend and divisor before division is attempted.
                        054A  1820
                     7E D4 054A  1821         CLRL    -(SP)                   ; Set aside space for U[0]
               52 6746 DE 054C  1822         MOVAL   (R7)[R6],R2             ; Retrieve dividend descriptor
                  52 62 7D 0550  1823         MOVQ    (R2),R2                 ; ... in two steps
                        0553  1824
                        0553  1825         MARK_POINT      DIVP_R6_R7
                  51 83 9A 0553  1826 30$:    MOVZBL  (R3)+,R1                ; Get next decimal digit pair
            7E 0000'CF41 9A 0556  1827         MOVZBL  DECIMAL$PACKED_TO_BINARY_TABLE[R1],-
                        055C  1828                         -(SP)               ; Convert digits to binary
                  F4 52 F5 055C  1829         SOBGTR  R2,30$                  ; Loop through entire input string
                        055F  1830
                        055F  1831 ; From this point until the common exit path for MULP and DIVP is entered,
                        055F  1832 ; no access violations that need to be backed out can occur. We do not need
                        055F  1833 ; to keep the address of ARITH_ACCVIO in R10 for this stretch of code. Note
                        055F  1834 ; that R10 must be reloaded before the exit code executes because the
                        055F  1835 ; destination string is written and may cause access violations.
                        055F  1836
               5A 6746 D0 055F  1837         MOVL    (R7)[R6],R10            ; Retrieve size of dividend array
                  5B 5E D0 0563  1838         MOVL    SP,R11                  ; R11 locates low order dividend digit
                        0566  1839
                        0566  1840 ; Allocate a scratch array on the stack the same size as the divisor array
                        0566  1841 ; (which is one larger than the number of digit pairs)
                        0566  1842
                  52 58 CE 0566  1843         MNEGL   R8,R2                   ; Need a negative index
            5E FC AE42 DE 0569  1844         MOVAL   -4(SP)[R2],SP           ; Adjust stack pointer
                        056E  1845
                        056E  1846 ;+
                        056E  1847 ; At this point, the stack and relevant general registers contain the
                        056E  1848 ; following information. In this description, N represents the number
                        056E  1849 ; of digit pairs in the divisor and M represents the number of digit
                        056E  1850 ; pairs in the dividend.
                        056E  1851 ;
                        056E  1852 ;                    +-------------------------------+ <-- SP
                        056E  1853 ;       scratch      |        N+1 longwords          |
                        056E  1854 ;                    +-------------------------------+ <-- R11
                        056E  1855 ;       dividend     |        M+1 longwords          |
                        056E  1856 ;                    +-------------------------------+ <-- R9
                        056E  1857 ;       divisor      |        N+1 longwords          |
                        056E  1858 ;                    +-------------------------------+ <-- R7
                        056E  1859 ;       quotient     |       M+1-N longwords         |
                        056E  1860 ;                    +-------------------------------+
                        056E  1861 ;                    |           R0..R11             |
                        056E  1862 ;                    +-------------------------------+
```

```
                                                I 10
                     056E  1863 ;
                     056E  1864 ;       R6  - Number of longwords in quotient array (M+1-N)
                     056E  1865 ;       R7  - Address of beginning of quotient array
                     056E  1866 ;       R8  - Number of digit pairs in divisor (called N)
                     056E  1867 ;       R9  - Address of low order digits in divisor
                     056E  1868 ;       R10 - Number of digit pairs in dividend (called M)
                     056E  1869 ;       R11 - Address of low order digits in dividend
                     056E  1870 ;-
                     056E  1871
          6E   DF    056E  1872          PUSHAL  (SP)                    ; Store address of scratch array
       7E 58   7D    0570  1873          MOVQ    R8,-(SP)                ; Remember divisor descriptor
       7E 5A   7D    0573  1874          MOVQ    R10,-(SP)               ; Remember dividend descriptor
                     0576  1875
                     0576  1876 ; The algorithm that guesses the quotient digit can be guaranteed to be off
                     0576  1877 ; by no more than two if the high order digit of the divisor (called V[1]) is
                     0576  1878 ; at least as large as 50 (our radix divided by 2). If the high order digit
                     0576  1879 ; is too small, we "normalize" the numerator and denomonator by multiplying
                     0576  1880 ; them by the same number, namely 100/(V[1]+1).
                     0576  1881
    FC A948    01    C1    0576  1882          ADDL3   #1,-4(R9)[R8],R0    ; Compute V[1] + 1
          50         0579
       33 50   D1    057C  1883          CMPL    R0,#51                  ; Compare to 50 + 1
          14   18    057F  1884          BGEQ    40$                     ; Skip normalization if V[1] big enough
          50   C7    0581  1885          DIVL3   R0,#100,R3              ; Compute normalization factor
 00000064    8F      0583
          53         0588
       54 58   7D    0589  1886          MOVQ    R8,R4                   ; Get descriptor of divisor
        00E0   30    058C  1887          BSBW    MULTIPLY_STRING         ; Normalize divisor
       54 5A   7D    058F  1888          MOVQ    R10,R4                  ; Get descriptor of dividend
        00DA   30    0592  1889          BSBW    MULTIPLY_STRING         ; Normalize dividend
                     0595  1890
                     0595  1891 ; We have now reached the point where we can start calculating quotient digits.
                     0595  1892 ; In the following loop, R5 and R6 are loop invariants. R5 contains the number
                     0595  1893 ; of digit pairs in the divisor. R6 always points to the longword beyond the
                     0595  1894 ; most significant digit in the dividend string. R7 and R8 must be loaded on
                     0595  1895 ; each pass through because these two pointers are modified. Notice that the
                     0595  1896 ; address of the divisor array is exactly what we want to store in R6.
                     0595  1897
       5A 56   7D    0595  1898 40$:    MOVQ    R6,R10                  ; Let R10/R11 describe quotient and loop
          5B   DD    0598  1899          PUSHL   R11                     ; Save quotient address for exit code
    5B 684A   DE    059A  1900          MOVAL   (R11)[R10],R11          ; Store quotient digits from high end
                     059E  1901
                     059E  1902 ; This rather harmless looking loop is where the work is done
                     059E  1903
       55 58   7D    059E  1904          MOVQ    R8,R5                   ; Initialize count and dividend address
       59 5A   D0    05A1  1905          MOVL    R10,R9                  ; Remember the loop count in R9
                     05A4  1906
    57   10 AE   7D    05A4  1907 50$:    MOVQ    16(SP),R7               ; Load divisor and scratch addresses
          001F   30    05A8  1908          BSBW    QUOTIENT_DIGIT          ; Get the next quotient digit
       7B 53   D0    05AB  1909          MOVL    R3,-(R11)               ; Store it
       56 04   C2    05AE  1910          SUBL    #4,R6                   ; "Advance" dividend pointer
         F0 5A   F5    05B1  1911          SOBGTR  R10,50$                 ; ... and go gack for more
                     05B4  1912
                     05B4  1913 ; The quotient digits have been stored on the stack. Eliminate the rest of the
                     05B4  1914 ; stack storage and enter the completion code that this routine shares with
                     05B4  1915 ; VAX$MULP. Note that R9 is already set up with the longword count use' by
                     05B4  1916 ; the exit code. Note also that R11 is pointing to the saved dividend descriptor
```

```
                                05B4  1917 ; that sits on top of the saved register array.
                                05B4  1918
                     5E   6E    D0 05B4  1919          MOVL    (SP),SP                 ; Reset stack pointer
                54   18 AB49    DE 05B7  1920          MOVAL   < <4*2> + -             ; Skip over saved dividend descriptor
                                05BC  1921                    <4*4> >(R11)[R9],R4     ;  and retrieve original R4 and R5
                     54   64    7D 05BC  1922          MOVQ    (R4),R4                 ; ... in two steps
                                05BF  1923
                                05BF  1924 ; The following is a HACK.
                                05BF  1925 ;
                                05BF  1926 ; The method used to obtain quotient digits generally leaves garbage (nonzero)
                                05BF  1927 ; in what will become the sign digit. (In fact, this is the tenths digit of a
                                05BF  1928 ; decimal expansion of the remainder.) We need to make the least significant
                                05BF  1929 ; digit a multiple of ten.
                                05BF  1930
                50   6E    0A    C7 05BF  1931          DIVL3   #10,(SP),R0             ; Divide by ten, losing remainder
                6E   50    0A    C5 05C3  1932          MULL3   #10,R0,(SP)             ; Store only tens digit
                                05C7  1933
                          FDB3    31 05C7  1934          BRW     MULTIPLY_DIVIDE_EXIT    ; Join common exit code
                                05CA  1935
                                05CA  1936          .DISABLE        LOCAL_BLOCK
```

```
05CA  1938          .SUBTITLE       QUOTIENT_DIGIT - Get Next Digit in Quotient
05CA  1939 ;+
05CA  1940 ; Functional Description:
05CA  1941 ;
05CA  1942 ;       This routine divides an (N+1)-element array of longwords by an N-element
05CA  1943 ;       array, producing a single quotient digit in the range of 0 to 99
05CA  1944 ;       inclusive. The dividend array is modified by subtracting the product
05CA  1945 ;       of the divisor array and the quotient digit.
05CA  1946 ;
05CA  1947 ;       The "numbers" that this array operates on multiple precision numbers
05CA  1948 ;       in radix 100. Each digit (a number between 0 and 99) is stored in a
05CA  1949 ;       longword array element with more significant digits stored at higher
05CA  1950 ;       addresses. The dividend string and the scratch string (also called the
05CA  1951 ;       product string) contain one more element than the divisor string.
05CA  1952 ;
05CA  1953 ; Input Parameters:
05CA  1954 ;
05CA  1955 ;       R5 - Number of "digits" (array elements) in divisor array (preserved)
05CA  1956 ;       R6 - Address of longword immediately following most significant
05CA  1957 ;            digit of dividend string (preserved)
05CA  1958 ;       R7 - Address of least significant digit in divisor string (modified)
05CA  1959 ;       R8 - Address of least significant digit in product string (modified)
05CA  1960 ;
05CA  1961 ; Output Parameters:
05CA  1962 ;
05CA  1963 ;       R3 - The quotient that results from dividing the dividend string
05CA  1964 ;            by the divisor string.
05CA  1965 ;
05CA  1966 ;       The final states of the three pointer registers are listed here
05CA  1967 ;       for completeness.
05CA  1968 ;
05CA  1969 ;       R6 - Address of longword immediately following most significant
05CA  1970 ;            digit of dividend string
05CA  1971 ;
05CA  1972 ;       R7 - Address of longword immediately following most significant digit
05CA  1973 ;            of divisor string. This longword must always contain zero.
05CA  1974 ;
05CA  1975 ;       R8 - Address of longword immediately following most significant
05CA  1976 ;            digit of product string
05CA  1977 ;
05CA  1978 ; Implicit Output:
05CA  1979 ;
05CA  1980 ;       The contents of the dividend array are modified to reflect the
05CA  1981 ;       subtraction of the product string. The result of this subtraction
05CA  1982 ;       could be stored elsewhere. It is a convenience to store it in the
05CA  1983 ;       dividend array on top of those array elements that are no longer
05CA  1984 ;       needed.
05CA  1985 ;
05CA  1986 ;       The contents of the divisor array are preserved.
05CA  1987 ;
05CA  1988 ; Side Effects:
05CA  1989 ;
05CA  1990 ;       R7 and R8 are modified by this routine. (See implicit output list.)
05CA  1991 ;
05CA  1992 ;       R5 and R6 are preserved.
05CA  1993 ;
05CA  1994 ;       R0, R1, R2, and R4 are used as scratch registers. R0 and R1 contain the
```

```
                                05CA  1995 ;       quadword result of EMUL that is then passed into EDIV  R2 is the
                                05CA  1996 ;       carry from one step tc the next. R4 is the loop counter.
                                05CA  1997 ;-
                                05CA  1998
                                05CA  1999 QUOTIENT_DIGIT:
  00000064 8F    7A            05CA  2000           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  2001           DIVL2   -4(R7)[R5],R0           ; R0 <- R0 / V[1]
      53    50   D0            05DA  2002           MOVL    R0,R3                   ; Store quotient "digit" in R3
            65   13            05DD  2003           BEQL    45$                     ; Nothing to do if quotient is zero
            53   D1            05DF  2004           CMPL    R3,#100                 ; Is quotient LEQU 99?
  00000064 8F                   05E1
            07   1F            05E6  2005           BLSSU   5$                      ; Branch if quotient OK
  00000063 8F    D0            05E8  2006           MOVL    #99,R3                  ; Otherwise start with 99
              53                05EE
                                05EF  2007
                                05EF  2008 ; We will now multiply the divisor array by the quotient digit, storing the
                                05EF  2009 ; product in the scratch array.
                                05EF  2010
            52   D4            05EF  2011 5$:      CLRL    R2                      ; Start out with a carry of zero
      54    55   D0            05F1  2012           MOVL    R5,R4                   ; R4 will be the loop counter
                                05F4  2013
52    87    53   7A            05F4  2014 10$:     EMUL    R3,(R7)+,R2,R0          ; Multiply next divisor digit
              50                05F8
  00000064 8F    7B            05F9  2015           EDIV    #100,R0,R2,(R8)+        ; Remainder to input array
88    52    50                  05FF
                                0602  2016                                       ; Quotient becomes carry
        EF 54    F5            0602  2017           SOBGTR  R4,10$                  ; More divisor digits?
                                0605  2018
      88    52   D0            0605  2019           MOVL    R2,(R8)+                ; Store final carry
                                0608  2020
                                0608  2021 ; If the product array is larger than the dividend array, then the quotient is
                                0608  2022 ; too large. To avoid a second trip through the rather costly EMUL/EDIV loop,
                                0608  2023 ; and also to avoid array subtraction that produces a negative result, we will
                                0608  2024 ; first compare the product and dividend arrays. If the product is smaller, we
                                0608  2025 ; can safely subtract. If the product is larger, we decrease the quotient by
                                0608  2026 ; one and subtract the divisor array from the product array.
                                0608  2027
50    56    D0                  0608  2028 15$:     MOVL    R6,R0                   ; Point R0 and R1 to high address ends
51    58    D0                  060B  2029           MOVL    R8,R1                   ; ... of dividend and scratch strings
54    55    D0                  060E  2030           MOVL    R5,R4                   ; Initialize the loop counter
                                0611  2031
                                0611  2032 ; The comparison is done from most to least significant digits
                                0611  2033
70    71    D1                  0611  2034 20$:     CMPL    -(R1),-(R0)             ; Compare next pair of digits
      0E    1F                  0614  2035           BLSSU   30$                     ; Leave loop if product is smaller
      2D    1A                  0616  2036           BGTRU   50$                     ; Also leave if product is larger
  F6  54    F4                  0618  2037           SOBGEQ  R4,20$                  ; More to test?
                                061B  2038
                                061B  2039 ; If we drop through the loop, then the dividend and product are equal. We
                                061B  2040 ; simply store zeros in the dividend array (the equivalent of subtraction
                                061B  2041 ; of equal arrays) and return. Note that R0 is already pointing to the
                                061B  2042 ; least significant dividend array element.
                                061B  2043
54    55    D0                  061B  2044           MOVL    R5,R4                   ; Initialize still another loop counter
                                061E  2045
```

```
                     80   D4  061E  2046 25$:    CLRL    (R0)+                   ; Store another zero
                  FB 54   F4  0620  2047         SOBGEQ  R4,25$                  ; Keep going?
                          0623  2048
                          05  0623  2049         RSB                             ; Return to caller
                          0624  2050
                          0624  2051 ; If we drop through the loop, then the quotient that is stored in R3 is good.
                          0624  2052 ; We need to subtract the product array from the dividend array. Note that R0
                          0624  2053 ; and R1 need to be adjusted to point to the least significant array elements
                          0624  2054 ; before the subtraction can begin.
                          0624  2055
                    54    54  CE  0624  2056 30$      MNEGL   R4,R4                  ; We need a negative index
                 50   6044  DE  0627  2057         MOVAL   (R0)[R4],R0             ; Adjust dividend pointer
                 51   6144  DE  062B  2058         MOVAL   (R1)[R4],R1             ; ... and product pointer
                    54    55  D0  062F  2059         MOVL    R5,R4                   ; R4 will count still another loop
                          0632  2060
                    80    81  C2  0632  2061 35$:    SUBL2   (R1)+,(R0)+             ; Subtract next digits
                          0A  18  0635  2062         BGEQ    40$                     ; Skip to end of loop if no borrow
                00000064  8F  C0  0637  2063         ADDL2   #100,-4(R0)             ; Add borrow back to this digit
                     FC   A0
                          60  D7  063F  2064         DECL    (R0)                    ; ... and borrow from next highest digit
                       EE 54  F4  0641  2065 40$:    SOBGEQ  R4,35$                  ; Keep going?
                          0644  2066
                          0644  2067 ; This is the exit path. R3 contains the quotient digit. The pointers to the
                          0644  2068 ; various input and scratch arrays are in an indeterminate state.
                          0644  2069
                          05  0644  2070 45$:    RSB                             ; Return to caller
                          0645  2071
                          0645  2072 ; The first guess at the quotient digit is too large. The brute force
                          0645  2073 ; approach is to decrement the quotient by one and execute the EMUL/EDIV loop
                          0645  2074 ; again. Note, however, that we can evaluate the modified product by
                          0645  2075 ; subtracting the divisor from the initial product. Note also that, because
                          0645  2076 ; the leading digit in the divisor is "large enough", we can only end up in
                          0645  2077 ; this code path twice. (That is, the initial guess at the quotient will
                          0645  2078 ; never be off by more than two.)
                          0645  2079
                       53   D7  0645  2080 50$:    DECL    R3                      ; Try quotient smaller by one
                       FB   13  0647  2081         BEQL    45$                     ; All done if zero
                          0649  2082
                          0649  2083 ; Point R1 and R2 at the least significant digits of the scratch and product
                          0649  2084 ; strings respectively.
                          0649  2085
                    50    55  CE  0649  2086         MNEGL   R5,R0                   ; Need a negative index
                51  FC A84C  DE  064C  2087         MOVAL   -4(R8)[R0],R1           ; Scratch array contains N+1 elements
                 52   6740  DE  0651  2088         MOVAL   (R7)[R0],R2             ; Product array contains N elements
                    54    55  D0  0655  2089         MOVL    R5,R4                   ; R4 will count still another loop
                          0658  2090
                    81    82  C2  0658  2091 60$:    SUBL2   (R2)+,(R1)+             ; Subtract next digits
                          0A  18  065B  2092         BGEQ    70$                     ; Skip to end of loop if no borrow
                00000064  8F  C0  065D  2093         ADDL2   #100,-4(R1)             ; Add borrow back to this digit
                     FC   A1
                          61  D7  0663  2094         DECL    (R1)                    ; ... and borrow from next highest digit
                       EE 54  F4  0665  2095 70$:    SOBGEQ  R4,60$                  ; Keep going?
                          0667  2096
                    51    04  C0  066A  2097         ADDL2   #4,R1                   ; Point R1 at most significant digit
                          99  11  066A  2098         BRB     15$                     ; Make another comparison
```

```
                        066F  2100          .SUBTITLE       MULTIPLY_STRING - Multiply a String by a Number
                        066F  2101 ;+
                        066F  2102 ; Functional Description:
                        066F  2103 ;
                        066F  2104 ;       This routine multiplies an array of numbers (each array element LEQU
                        066F  2105 ;       99) by a number (also LEQU 99). Each array element in the input array
                        066F  2106 ;       is replaced with the modified product, with the carry propogated to
                        066F  2107 ;       the next array element.
                        066F  2108 ;
                        066F  2109 ; Input Parameters:
                        066F  2110 ;
                        066F  2111 ;       R3 - Multiplier
                        066F  2112 ;       R4 - Input array size
                        066F  2113 ;       R5 - Input array address
                        066F  2114 ;
                        066F  2115 ; Output Parameters:
                        066F  2116 ;
                        066F  2117 ;       None
                        066F  2118 ;
                        066F  2119 ; Implicit Output:
                        066F  2120 ;
                        066F  2121 ;       The input array elements are altered.
                        066F  2122 ;
                        066F  2123 ; Side Effects:
                        066F  2124 ;
                        066F  2125 ;       R4 and R5 are modified by this routine.
                        066F  2126 ;
                        066F  2127 ;       R3 is preserved.
                        066F  2128 ;
                        066F  2129 ;       R0, R1, and R2 are used as scratch registers. R0 and R1 contain the
                        066F  2130 ;       quadword result of EMUL that is then passed into EDIV. R2 is the
                        066F  2131 ;       carry from one step to the next.
                        066F  2132 ;
                        066F  2133 ; Assumptions:
                        066F  2134 ;
                        066F  2135 ;       This routine assumes that all array elements lie in the range from 0
                        066F  2136 ;       to 99 inclusive. (This is true if all input strings contain only legal
                        066F  2137 ;       decimal digits.) The arithmetic performed by this routine will
                        066F  2138 ;       maintain this assumption. The details of this argument can be found in
                        066F  2139 ;       the routine header for EXTENDED_MULTIPLY_STRING. This routine performs
                        066F  2140 ;       less work so that those arguments also apply here.
                        066F  2141 ;-
                        066F  2142
                        066F  2143 MULTIPLY_STRING:
            52    D4    066F  2144          CLRL    R2                      ; Initial carry is zero
                        0671  2145
52  65      53    7A    0671  2146 10$:     EMUL    R3,(R5),R2,R0           ; Form modified product (R0 LEQU 9900)
                  50    0675
 00000064   8F    7B    0676  2147          EDIV    #100,R0,R2,(R5)+        ; Remainder to input array
85  52      50          067C
                        067F  2148                                          ; Quotient becomes carry
        EF  54    F5    067F  2149          SOBGTR  R4,10$                  ; Keep going?
                        0682  2150
      65    52    D0    0682  2151          MOVL    R2,(R5)                 ; Store final carry
                  05    0685  2152          RSB
```

```
                                                B 11
0686  2154                .SUBTITLE       DECIMAL_ROPRAND
0686  2155 ;-
0686  2156 ; Functional Description:
0686  2157 ;
0686  2158 ;       This routine receives control when a digit count larger than 31
0686  2159 ;       is detected. The exception is architecturally defined as an
0686  2160 ;       abort so there is no need to store intermediate state. All of the
0686  2161 ;       routines in this module save all registers R0 through R11 before
0686  2162 ;       performing the digit check. These registers must be restored
0686  2163 ;       before control is passed to VAX$ROPRAND.
0686  2164 ;
0686  2165 ; Input Parameters:
0686  2166 ;
0686  2167 ;       00(SP) - Saved R0
0686  2168 ;         .
0686  2169 ;         .
0686  2170 ;       44(SP) - Saved R11
0686  2171 ;       48(SP) - Return PC from VAX$xxxxxx routine
0686  2172 ;
0686  2173 ; Output Parameters:
0686  2174 ;
0686  2175 ;       00(SP) - Offset in packed register array to delta PC byte
0686  2176 ;       04(SP) - Return PC from VAX$xxxxxx routine
0686  2177 ;
0686  2178 ; Implicit Output:
0686  2179 ;
0686  2180 ;       This routine passes control to VAX$ROPRAND where further
0686  2181 ;       exception processing takes place.
0686  2182 ;-
0686  2183
0686  2184                ASSUME ADDP6_B_DELTA_PC EQ ADDP4_B_DELTA_PC
0686  2185                ASSUME SUBP4_B_DELTA_PC EQ ADDP4_B_DELTA_PC
0686  2186                ASSUME SUBP6_B_DELTA_PC EQ ADDP4_B_DELTA_PC
0686  2187                ASSUME MULP_B_DELTA_PC  EQ ADDP4_B_DELTA_PC
0686  2188                ASSUME DIVP_B_DELTA_PC  EQ ADDP4_B_DELTA_PC
0686  2189
0686  2190 DECIMAL_ROPRAND:
OFFF 8F   BA  0686  2191        POPR    #^M<R0,R1,R2,R3,R4,R5,R6,R7,R8,R9,R10,R11>
   03     DD  068A  2192        PUSHL   #ADDP4_B_DELTA_PC       ; Store offset to delta PC byte
 F971'    31  068C  2193        BRW     VAX$ROPRAND             ; Pass control along
```

```
                                        .SUBTITLE       ARITH_ACCVIO - Reflect an Access Violation
                                ;+
                                ; Functional Description:
                                ;
                                ;       This routine receives control when an access violation occurs while
                                ;       executing within the emulator routines for ADDP4, ADDP6, SUBP4, SUBP6,
                                ;       MULP, or DIVP.
                                ;
                                ;       The routine header for ASHP_ACCVIO in module VAX$ASHP contains a
                                ;       detailed description of access violation handling for the decimal
                                ;       string instructions.
                                ;
                                ; Input Parameters:
                                ;
                                ;       See routine ASHP_ACCVIO in module VAX$ASHP
                                ;
                                ; Output Parameters:
                                ;
                                ;       See routine ASHP_ACCVIO in module VAX$ASHP
                                ;-
                                ;
                                ARITH_ACCVIO:
                        52  D4 068F 2217           CLRL    R2                      ; Initialize the counter
                    F96B CF  9F 0691 2218           PUSHAB  MODULE_BASE             ; Store base address of this module
                    06FC'CF  9F 0695 2219           PUSHAB  MODULE_END              ; Store module end address
                       F964' 30 0699 2220           BSBW    DECIMA[$BOUNDS_CHECK    ; Check if PC is inside the module
                    5E  04  C0 069C 2221           ADDL    #4,SP                   ; Discard end address
                    51  8E  C2 069F 2222           SUBL2   (SP)+,R1                ; Get PC relative to this base
                                06A2 2223
                0000'CF42   51  B1 06A2 2224 10$:   CMPW    R1,PC_TABLE_BASE[R2]    ; Is this the right PC?
                            07  13 06A8 2225           BEQL    30$                     ; Exit loop if true
                        F4 52   2A F2 06AA 2226           AOBLSS  #TABLE_SIZE,R2,10$      ; Do the entire table
                                06AE 2227
                                06AE 2228 ; If we drop through the dispatching based on PC, then the exception is not
                                06AE 2229 ; one that we want to back up. We simply reflect the exception to the user.
                                06AE 2230
                            0F  BA 06AE 2231 20$:   POPR    #^M<R0,R1,R2,R3>        ; Restore saved registers
                            05  05 0680 2232           RSB                             ; Return to exception dispatcher
                                06B1 2233
                                06B1 2234 ; The exception PC matched one of the entries in our PC table. R2 contains
                                06B1 2235 ; the index into both the PC table and the handler table. R1 has served
                                06B1 2236 ; its purpose and can be used as a scratch register.
                                06B1 2237
                51  0000'CF42   3C 06B1 2238 30$:   MOVZWL  HANDLER_TABLE_BASE[R2],R1       ; Get the offset to the handler
                    F944 CF41   17 06B7 2239           JMP     MODULE_BASE[R1]             , Pass control to the handler
                                06BC 2240
                                06BC 2241 ; In all of the instruction-specific routines, the state of the stack
                                06BC 2242 ; will be shown as it was when the exception occurred. All offsets will
                                06BC 2243 ; be pictured relative to R0.
```

```
                     06BC  2245          .SUBTITLE       Access Violation Handling for ADDPx and SUBPx
                     06BC  2246 ;+
                     06BC  2247 ; Functional Description.
                     06BC  2248 ;
                     06BC  2249 ;       The only difference among the various entry points is the number of
                     06BC  2250 ;       longwords on the stack. R0 is advanced beyond these longwords to point
                     06BC  2251 ;       to the list of saved registers. These registers are then restored,
                     06BC  2252 ;       effectively backing the routine up to its initial state.
                     06BC  2253 ;
                     06BC  2254 ; Input Parameters:
                     06BC  2255 ;
                     06BC  2256 ;       R0 - Address of top of stack when access violation occurred
                     06BC  2257 ;
                     06BC  2258 ;       See specific entry points for details
                     06BC  2259 ;
                     06BC  2260 ; Output Parameters:
                     06BC  2261 ;
                     06BC  2262 ;       See input parameter list for VAX$DECIMAL_ACCVIO in module VAX$ASHP
                     06BC  2263 ;-
                     06BC  2264
                     06BC  2265 ;+
                     06BC  2266 ; ADD_SUB_BSBW_24
                     06BC  2267 ;
                     06BC  2268 ; An access violation occurred in one of the subroutines ADD_PACKED_BYTE,
                     06BC  2269 ; SUB_PACKED_BYTE, or STORE_RESULT. In addition to the six longwords of work
                     06BC  2270 ; space, this routine has an additional longword, the return PC, on the
                     06BC  2271 ; stack.
                     06BC  2272 ;
                     06BC  2273 ;       00(R0) - Return PC in mainline VAX$xxxxxx routine
                     06BC  2274 ;       04(R0) - Address of sign byte of destination string
                     06BC  2275 ;       08(R0) - First longword of scratch space
                     06BC  2276 ;        etc.
                     06BC  2277 ;-
                     06BC  2278
                     06BC  2279 ADD_SUB_BSBW_24:
 50    04    C0      06BC  2280          ADDL    #4,R0                   ; Skip over return PC and drop into ...
                     06BF  2281
                     06BF  2282 ;+
                     06BF  2283 ; ADD_SUB_24
                     06BF  2284 ;
                     06BF  2285 ; There are five longwords of workspace and a saved string address on the stack
                     06BF  2286 ; for this entry point.
                     06BF  2287 ;
                     06BF  2288 ;       00(R0) - Address of sign byte of destination string
                     06BF  2289 ;       04(R0) - First longword of scratch space
                     06BF  2290 ;          .
                     06BF  2291 ;          .
                     06BF  2292 ;       20(R0) - Fifth longword of scratch space
                     06BF  2293 ;       24(SP) - Saved R0
                     06BF  2294 ;       28(SP) - Saved R1
                     06BF  2295 ;        etc.
                     06BF  2296 ;-
                     06BF  2297
                     06BF  2298 ADD_SUB_24:
 50    18    C0      06BF  2299          ADDL    #24,R0                  ; Discard scratch space on stack
    F93B'    31      06C2  2300          BRW     VAX$DECIMAL_ACCVIO      ; Join common code to restore registers
                     06C5  2301
```

```
                                E 11
                                06C5  2302 ;+
;LJK0045                        06C5    .1 ; ADD_SUB_BSBW_4
;LJK0045                        06C5    .2 ;
;LJK0045                        06C5    .3 ; An access violation occurred in the subroutine CHECK_WRITE_ACCESS. This
;LJK0045                        06C5    .4 ; entry point has two additional longwords, a copy of the saved R4 and the
;LJK0045                        06C5    .5 ; return PC, on the stack on top of the saved register array.
;LJK0045                        06C5    .6 ;
;LJK0045                        06C5    .7 ;       00(R0) - Saved R4
;LJK0045                        06C5    .8 ;       04(R0) - Return PC in mainline VAX$xxxP4 routine
;LJK0045                        06C5    .9 ;       08(R0) - Saved R0
;LJK0045                        06C5   .10 ;       12(R0) - Saved R1
;LJK0045                        06C5   .11 ;        etc.
;LJK0045                        06C5   .12 ;-
;LJK0045                        06C5   .13
;LJK0045                        06C5   .14 ADD_SUB_BSBW_4:
;LJK0045          50    04   C0  06C5   .15         ADDL    #4,R0                   ; Skip over saved R4 and drop into ...
;LJK0045                        06C8   .16
;LJK0045                        06C8   .17 ;+
                                06C8  2303 ; ADD_SUB_BSBW_0
                                06C8  2304 ;
                                06C8  2305 ; An access violation occurred in one of the subroutine STRIP_ZEROS. This
                                06C8  2306 ; entry point has an additional longword, the return PC, on the stack on top
                                06C8  2307 ; of the saved register array.
                                06C8  2308 ;
                                06C8  2309 ;       00(R0) - Return PC in mainline VAX$xxxxxx routine
                                06C8  2310 ;       04(R0) - Saved R0
                                06C8  2311 ;       08(R0) - Saved R1
                                06C8  2312 ;        etc.
                                06C8  2313 ;-
                                06C8  2314
                                06C8  2315 ADD_SUB_BSBW_0:
          50    04   C0  06C8  2316         ADDL    #4,R0                   ; Skip over return PC and ...
             F932'   31  06CB  2317         BRW     VAX$DECIMAL_ACCVIO      ; Join common code to restore registers
```

```
                     06CE  2319          .SUBTITLE       Access Violation Handling for MULP and DIVP
                     06CE  2320 ;+
                     06CE  2321 ; Functional Description:
                     06CE  2322 ;
                     06CE  2323 ;       The only difference among the various entry points is the number of
                     06CE  2324 ;       longwords on the stack. R0 is advanced beyond these longwords to point
                     06CE  2325 ;       to the list of saved registers. These registers are then restored,
                     06CE  2326 ;       effectively backing the routine up to its initial state.
                     06CE  2327 ;
                     06CE  2328 ; Input Parameters:
                     06CE  2329 ;
                     06CE  2330 ;       R0 - Address of top of stack when access violation occurred
                     06CE  2331 ;
                     06CE  2332 ;       See specific entry points for details
                     06CE  2333 ;
                     06CE  2334 ; Output Parameters:
                     06CE  2335 ;
                     06CE  2336 ;       See input parameter list for VAX$DECIMAL_ACCVIO in module VAX$ASHP
                     06CE  2337 ;-
                     06CE  2338
                     06CE  2339 ;+
                     06CE  2340 ; MULP_R8
                     06CE  2341 ;
                     06CE  2342 ; An access violation occurred while MULP was accessing one of its two source
                     06CE  2343 ; strings. In this particular case, MULP was storing the longer of the two
                     06CE  2344 ; input strings in a longword array on the top of the stack. There is an
                     06CE  2345 ; array of R8 longwords on top of an array of 32 longwords on top of the
                     06CE  2346 ; saved register array.
                     06CE  2347 ;
                     06CE  2348 ;       R8 - Number of longwords on top of the 32-longword array
                     06CE  2349 ;-
                     06CE  2350
                     06CE  2351          .ENABLE         LOCAL_BLOCK
                     06CE  2352
                     06CE  2353 MULP_R8:
          50   6048  DE  06CE  2354          MOVAL   (R0)[R8],R0             ; Discard input array storage
               04   11  06D2  2355          BRB     10$                     ; Might as well share a little code
                     06D4  2356
                     06D4  2357 ;+
                     06D4  2358 ; MULP_AT_SP
                     06D4  2359 ;
                     06D4  2360 ; An access violation occurred while MULP was accessing one of its two source
                     06D4  2361 ; strings. In this case, the access violation occurred in the middle of the
                     06D4  2362 ; grand multiply loop as a digit pair was being retrieved from the shorter of
                     06D4  2363 ; the two input strings. The address of the start of the 32-longword array
                     06D4  2364 ; was itself stored on top of the stack for convenience.
                     06D4  2365 ;
                     06D4  2366 ;       00(R0) - Saved byte count of longer input string
                     06D4  2367 ;       04(R0) - Saved address of longer input string
                     06D4  2368 ;       08(R0) - Address of 32-longword array farther down the stack
                     06D4  2369 ;-
                     06D4  2370
                     06D4  2371 MULP_AT_SP:
       50   08 A0  D0  06D4  2372          MOVL    8(R0),R0                ; Locate start of 32-longword array
  50   0088 C0  9E  06D8  2373 10$:       MOVAB   <<4*32> + <4*2>>(R0),R0 ; Throw that away, too
            F920'  31  06DD  2374          BRW     VAX$DECIMAL_ACCVIO      ; Join common code to restore registers
                     06E0  2375
```

```
                    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  ;+
```

```
                                                H 11
                     06EE  2433 ; MULP_BSBW_0
                     06EE  2434 ; DIVP_BSBW_0
                     06EE  2435 ;
                     06EE  2436 ; An access violation occurred in one of the subroutine STRIP_ZEROS. This
                     06EE  2437 ; entry point has an additional longword, the return PC, on the stack on top
                     06EE  2438 ; of the saved register array.
                     06EE  2439 ;
                     06EE  2440 ;       00(R0) - Return PC in mainline VAX$MULP or VAX$DIVP routine
                     06EE  2441 ;       04(R0) - Saved R0
                     06EE  2442 ;       08(R0) - Saved R1
                     06EE  2443 ;        etc.
                     06EE  2444 ;-
                     06EE  2445
                     06EE  2446 MULP_BSBW_0:
                     06EE  2447 DIVP_BSBW_0:
           50   04   C0  06EE  2448         ADDL    #4,R0                   ; Skip over return PC and drop into ...
                     06F1  2449
                     06F1  2450 ;+
                     06F1  2451 ; DIVP_0
                     06F1  2452 ; MULP_DIVP_0
                     06F1  2453 ;
                     06F1  2454 ; There was nothing allocated on the stack other than the saved register
                     06F1  2455 ; array when the access violation occurred. We merely pass control to common
                     06F1  2456 ; code to restore the registers.
                     06F1  2457 ;
                     06F1  2458 ;       00(R0) - Saved R0
                     06F1  2459 ;       04(R0) - Saved R1
                     06F1  2460 ;        etc.
                     06F1  2461 ;-
                     06F1  2462
                     06F1  2463 DIVP_0:
                     06F1  2464 MULP_DIVP_0:
     F90C'   31      06F1  2465         BRW     VAX$DECIMAL_ACCVIO      ; Join common code to restore registers
                     06F4  2466
                     06F4  2467 ;+
                     06F4  2468 ; DIVP_R6_R7
                     06F4  2469 ;
                     06F4  2470 ; An access violation occurred while one of the two input strings was being
                     06F4  2471 ; converted to an array of longwords on the stack. The state of the stack
                     06F4  2472 ; is rather complicated but R6 and R7 contain enough information to allow
                     06F4  2473 ; the rest of the stack contents to be ignored.
                     06F4  2474 ;
                     06F4  2475 ;       R6 - Count of longwords in quotient array on stack
                     06F4  2476 ;       R7 - Address of quotient array on stack
                     06F4  2477 ;
                     06F4  2478 ;       00(R0) - First longword of quotient array
                     06F4  2479 ;         .
                     06F4  2480 ;         .
                     06F4  2481 ;       zz-4(R0) - Last longword of scratch storage
                     06F4  2482 ;       zz+0(R0) - Digit count of dividend string
                     06F4  2483 ;       zz+4(R0) - Address of dividend string
                     06F4  2484 ;       zz+8(R0) - Saved R0
                     06F4  2485 ;       zz+12(R0) - Saved R1
                     06F4  2486 ;        etc.
                     06F4  2487 ;
                     06F4  2488 ;       where zz = 4 * R6
                     06F4  2489 ;-
```

```
                                                                I 11
                    06F4  2490
                    06F4  2491 DIVP_R6_R7:
          50  08 A746   DE  06F4  2492          MOVAL   8(R7)[R6],R0            ; Discard everything on stack
              F904'     31  06F9  2493          BRW     VAX$DECIMAL_ACCVIO      ; Join common code to restore registers
                    06FC  2494
                    06FC  2495          END_MARK_POINT
                    06FC  2496
                    06FC  2497          .END
```

```
...PC...                          = 00000553
...ROPRAND...                     = 000004BF  R      02
ADDP4_B_DELTA_PC                  = 00000003
ADDP6_B_DELTA_PC                  = 00000003
ADD_PACKED                          0C0000E5  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$V_Z                           = 00000002
QUOTIENT_DIGIT                      000005CA  R      02
SRMSK_FLT_DIV_T                   = 00000004
STORE_RESULT                        00000254  R      02
SUBP4_B_DELTA_PC                  = 00000003
SUBP6_B_DELTA_PC                  = 00000003
SUBTRACT_PACKED                     00000198  R      02
SUB_PACKED_BYTE_R6_R7               0000022E  R      02
SUB_PACKED_BYTE_STRING              00000228  R      02
TABLE_SIZE                        = 0000002A
VAX$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 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
_255DUA18:[EMULAT.OBJ]VAXMACROS.MLB;1	12
_255DUA18:[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

ERF
MAP

REGSELECT
LIS