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

FILEID**VAXDECIML

```
VV      VV    AAAAAA    XX      XX  DDDDDDDD    EEEEEEEEEE    CCCCCCCC    IIIIII    MM      MM  LL
VV      VV    AAAAAA    XX      XX  DDDDDDDD    EEEEEEEEEE    CCCCCCCC    IIIIII    MM      MM  LL
VV      VV  AA      AA  XX      XX  DD      DD  EE          CC              II      MMMM  MMMM  LL
VV      VV  AA      AA  XX      XX  DD      DD  EE          CC              II      MMMM  MMMM  LL
VV      VV  AA      AA    XX  XX    DD      DD  EE          CC              II      MM  MM  MM  LL
VV      VV  AA      AA    XX  XX    DD      DD  EE          CC              II      MM  MM  MM  LL
VV      VV  AA      AA      XX      DD      DD  EEEEEEEE    CC              II      MM      MM  LL
VV      VV  AA      AA      XX      DD      DD  EEEEEEEE    CC              II      MM      MM  LL
VV      VV  AAAAAAAAAA    XX  XX    DD      DD  EE          CC              II      MM      MM  LL
VV      VV  AAAAAAAAAA    XX  XX    DD      DD  EE          CC              II      MM      MM  LL
  VV  VV    AA      AA  XX      XX  DD      DD  EE          CC              II      MM      MM  LL              ....
  VV  VV    AA      AA  XX      XX  DD      DD  EE          CC              II      MM      MM  LL              ....
    VV      AA      AA  XX      XX  DDDDDDDD    EEEEEEEEEE    CCCCCCCC    IIIIII    MM      MM  LLLLLLLLLL      ....
    VV      AA      AA  XX      XX  DDDDDDDD    EEEEEEEEEE    CCCCCCCC    IIIIII    MM      MM  LLLLLLLLLL      ....

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

(2)	87	Miscellaneous Notes
(5)	242	Declarations
(6)	277	Conversion Tables
(7)	402	VAX$CMPPx - Compare Packed
(7)	457	Data Declarations for CMPP3 and CMPP4
(8)	694	VAX$MOVP - Move Packed
(9)	808	Routine to Strip Leading Zeros from Decimal String
(10)	947	DECIMAL_ROPRAND
(11)	980	DECIMAL_ACCVIO - Reflect an Access Violation
(12)	1026	Context-Specific Access Violation Handling for VAX$CMPPx
(13)	1089	Context-Specific Access Violation Handling for VAX$MOVP

```
0000        .TITLE  VAX$DECIMAL - VAX-11 Packed Decimal Instruction Emulator
0000        .IDENT  /V04-000/
0000
0000  ;
0000  ;*************************************************************************
0000  ;*                                                                       *
0000  ;*  COPYRIGHT (c) 1978, 1980, 1982, 1984 BY                              *
0000  ;*  DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.               *
0000  ;*  ALL RIGHTS RESERVED.                                                 *
0000  ;*                                                                       *
0000  ;*  THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED *
0000  ;*  ONLY IN  ACCORDANCE WITH  THE  TERMS  OF  SUCH  LICENSE  AND WITH THE *
0000  ;*  INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR  ANY  OTHER *
0000  ;*  COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY *
0000  ;*  OTHER PERSON.  NO TITLE TO AND OWNERSHIP OF  THE  SOFTWARE IS  HEREBY *
0000  ;*  TRANSFERRED.                                                         *
0000  ;*                                                                       *
0000  ;*  THE INFORMATION IN THIS SOFTWARE IS  SUBJECT TO CHANGE WITHOUT NOTICE *
0000  ;*  AND  SHOULD  NOT  BE  CONSTRUED AS  A COMMITMENT BY DIGITAL EQUIPMENT *
0000  ;*  CORPORATION.                                                         *
0000  ;*                                                                       *
0000  ;*  DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE  OR  RELIABILITY OF ITS *
0000  ;*  SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.              *
0000  ;*                                                                       *
0000  ;*                                                                       *
0000  ;*************************************************************************
0000  ;
0000
0000  ;++
0000  ; Facility:
0000  ;
0000  ;       VAX-11 Instruction Emulator
0000  ;
0000  ; Abstract:
0000  ;
0000  ;       The routines in this module emulate the VAX-11 packed decimal
0000  ;       instructions. These procedures can be a part of an emulator
0000  ;       package or can be called directly after the input parameters
0000  ;       have been loaded into the architectural registers.
0000  ;
0000  ;       The input parameters to these routines are the registers that
0000  ;       contain the intermediate instruction state.
0000  ;
0000  ; Environment:
0000  ;
0000  ;       These routines run at any access mode, at any IPL, and are AST
0000  ;       reentrant.
0000  ;
0000  ; Author:
0000  ;
0000  ;       Lawrence J. Kenah
0000  ;
0000  ; Creation Date
0000  ;
0000  ;       24 September 1982
0000  ;
0000  ; Modified by:
```

```
0000    ;
0000    ;       V01-007 LJK0043         Lawrence J. Kenah       26-Jul-1984
0000    ;               Change STRIP_ZEROS routines so that they are more forgiving
0000    ;               when confronted with poorly formed packed decimal strings.
0000    ;               Specifically, do not allow string lengths smaller than zero.
0000    ;
0000    ;       V01-006 LJK0038         Lawrence J. Kenah       19-Jul-1984
0000    ;               Insure that initial setting of C-bit is preserved when
0000    ;               MOVP is restarted after an access violation.
0000    ;
0000    ;       V01-005 LJK0024         Lawrence J. Kenah       20-Feb-1984
0000    ;               Add code to handle access violations.
0000    ;
0000    ;       V01-004 LJK0013         Lawrence J. Kenah       17-Nov-1983
0000    ;               Move CVTPL to a separate module.
0000    ;
0000    ;       V01-003 LJK0008         Lawrence J. Kenah       18-Oct-1983
0000    ;               Move decimal arithmetic and numeric string routines to
0000    ;               separate modules.
0000    ;
0000    ;       V01-002 LJK0006         Lawrence J. Kenah       14-Oct-1983
0000    ;               Fix code that handles arithmetic traps. Add reserved operand
0000    ;               processing. Add PROBEs and other code to handle access
0000    ;               violations.
0000    ;
0000    ;       V01-001 Original        Lawrence J. Kenah       24-Sep-82
0000    ;
0000    ;--
```

```
0000          .SUBTITLE       Miscellaneous Notes
0000
0000  ;+
0000  ; There are several techniques that are used throughout the routines in this
0000  ; module that are worth a comment somewhere. Rather than duplicate near
0000  ; identical commentary in several places, we will describe these general
0000  ; techniques in a single place.
0000  ;
0000  ; 1.     The VAX-11 architecture specifies that several kinds of input produce
0000  ;        UNPREDICTABLE results. They are:
0000  ;
0000  ;         o  Illegal decimal digit in packed decimal string
0000  ;
0000  ;         o  Illegal sign specifier (other than 10 through 15) in low nibble of
0000  ;            highest addressed byte of packed decimal string
0000  ;
0000  ;         o  Packed decimal string with even number of digits that contains
0000  ;            other than a zero in the high nibble of the lowest addressed byte
0000  ;
0000  ;        These routines take full advantage of the meaning of UNPREDICTABLE.
0000  ;        In general, the code assumes that all input is correct. The operation
0000  ;        of the code for illegal input is not even consistent but is simply
0000  ;        whatever happens to be convenient in a particular place.
0000  ;
0000  ; 2.     All of these routines accumulate information about condition codes at
0000  ;        several key places in a routine. This information is kept in a
0000  ;        register (usually R11) that is used to set the final condition codes
0000  ;        in the PSW. In order to allow the register to obtain its correct
0000  ;        contents when the routine exits (without further affecting the
0000  ;        condition codes), the condition codes are set from the register
0000  ;        (BISPSW reg) and the register is then restored with a POPR
0000  ;        instruction, which does not affect condition codes.
0000  ;
0000  ; 3.     There are several instances in these routines where it is necessary to
0000  ;        determine the difference in length between an input and an output
0000  ;        string and perform special processing on the excess digits. When the
0000  ;        longer string is a packed decimal string (it does not matter if the
0000  ;        packed decimal string is an input string or an output string), it is
0000  ;        sometimes useful to convert the difference in digits to a byte count.
0000  ;
0000  ;        There are four different cases that exist. We will divide these cases
0000  ;        into two sets of two cases, depending on whether the shorter length is
0000  ;        even or odd.
0000  ;
0000  ;        In the pictures that appear below, a blank box indicates a digit in
0000  ;        the shorter string. A string of three dots in a box indicates a digit
0000  ;        in the longer string. A string of three stars indicates an unused
0000  ;        digit in a decimal string. The box that contains +/- obviously
0000  ;        indicates the sign nibble in a packed decimal string.
0000  ;
0000  ; (cont.)
```

M 5

```
0000 ;                                                      +--------+--------+
0000 ;                                                      |        |        |
0000 ;                                                      |  ***   |  ...   |
0000 ;                                                      |        |        |
0000 ;                  +--------+--------+                 +--------+ - - - -+
0000 ;                  |        |        |                 |        |        |
0000 ;                  |  ...   |  ...   |                 |  ...   |  ...   |
0000 ;                  |        |        |                 |        |        |
0000 ;                  + - - - -+--------+                 + - - - -+--------+
0000 ;                  |        |                          |        |
0000 ;                  |  ...   |                          |  ...   |
0000 ;                  |        |                          |        |
0000 ;                  +--------+ - - - -+                 +--------+ - - - -+
0000 ;                  |        |        |                 |        |        |
0000 ;                  |        |        |                 |        |        |
0000 ;                  |        |        |                 |        |        |
0000 ;                  + - - - -+ - - - -+                 + - - - -+ - - - -+
0000 ;                  |        |        |                 |        |        |
0000 ;                  |        |  +/-   |                 |        |  +/-   |
0000 ;                  |        |        |                 |        |        |
0000 ;                  +--------+--------+                 +--------+--------+
0000 ;
0000 ;               A  Longer string odd                B  Longer string even
0000 ;                  Difference odd                      Difference even
0000 ;
0000 ;
0000 ;                     CASE 1  Shorter string has even number of digits
0000 ;
0000 ;
0000 ;
0000 ;                  +--------+--------+                 +--------+--------+
0000 ;                  |        |        |                 |        |        |
0000 ;                  |  ...   |  ...   |                 |  ***   |  ...   |
0000 ;                  |        |        |                 |        |        |
0000 ;                  + - - - -+ - - - -+                 +--------+ - - - -+
0000 ;                  |        |        |                 |        |        |
0000 ;                  |  ...   |  ...   |                 |  ...   |  ...   |
0000 ;                  |        |        |                 |        |        |
0000 ;                  +--------+--------+                 +--------+--------+
0000 ;                  |        |        |                 |        |        |
0000 ;                  |        |        |                 |        |        |
0000 ;                  |        |        |                 |        |        |
0000 ;                  + - - - -+ - - - -+                 + - - - -+ - - - -+
0000 ;                  |        |        |                 |        |        |
0000 ;                  |        |  +/-   |                 |        |  +/-   |
0000 ;                  |        |        |                 |        |        |
0000 ;                  +--------+--------+                 +--------+--------+
0000 ;
0000 ;               A  Longer string odd                B  Longer string even
0000 ;                  Difference even                     Difference odd
0000 ;
0000 ;
0000 ;                      CASE 2  Shorter string has odd number of digits
0000 ;
0000 ; (cont.)
```

```
0000 ;
0000 ;        In general, the code must calculate the number of bytes that contain
0000 ;        the excess digits. Most of the time, the interesting number includes
0000 ;        complete excess bytes. The excess digit in the high nibble of the
0000 ;        highest addressed byte (both parts of Case 1) is ignored.
0000 ;
0000 ;        In three out of four cases, the difference (called R5 from this point
0000 ;        on) can be simply divided by two to obtain a byte count. In one case
0000 ;        (Case 2 B), this is not correct. (For example, 3/2 = 1 and we want to
0000 ;        get a result of 2.) Note, however, that in both parts of Case 2, we
0000 ;        can add 1 to R5 before we divide by two. In Case 2 B, this causes the
0000 ;        result to be increased by 1, which is what we want. In Case 2 A,
0000 ;        because the original difference is even, an increment of one before we
0000 ;        divide by two has no effect on the final result.
0000 ;
0000 ;        The correct code sequence to distinguish case 2 B from the other three
0000 ;        cases involves two BLBx instructions. A simpler sequence that
0000 ;        accomplishes correct results in all four cases when converting a digit
0000 ;        count to a byte count is something like
0000 ;
0000 ;                        BLBC    length-of-shorter,10$
0000 ;                        INCL    R5
0000 ;                10$:    ASHL    #-1,R5,R5
0000 ;
0000 ;        where the length of the shorter string will typically be contained in
0000 ;        either R0 or R2.
0000 ;
0000 ;        Note that we could also look at both B parts, performing the extra
0000 ;        INCL instruction when the longer string is even. In case 1 B, this
000C ;        increment transforms an even difference to an odd number but does not
0000 ;        affect the division by two. In case 2 B, the extra increment produces
0000 ;        the correct result. This option is not used in these routines.
0000 ;
0000 ;        The two routines for CVTSP and CVTTP need a slightly different number.
0000 ;        They want the number of bytes including the byte containing the excess
0000 ;        high nibble. For Case 2, the above calculation is still valid. For
0000 ;        Case 1, it is necessary to add one to R5 after the R5 is divided by
0000 ;        two to obtain the correct byte count.
0000 ;
0000 ; 4.     There is a routine called STRIP_ZEROS that removes high order zeros
0000 ;        from decimal strings. This routine is not used by all of the routines
0000 ;        in this module but only by those routines that perform complicated
0000 ;        calculations on each byte of the input string. For these routines, the
0000 ;        overhead of testing for and discarding leading zeros is less than the
0000 ;        more costly per byte overhead of these routines.
0000 ;-
```

```
    0000          .SUBTITLE       Declarations
    0000
    0000  ; Include files:
    0000
    0000          .NOCROSS                        ; No cross reference for these
    0000          .ENABLE         SUPPRESSION     ; No symbol table entries either
    0000
    0000          CMPP3_DEF                       ; Bit fields in CMPP3 registers
    0000          CMPP4_DEF                       ; Bit fields in CMPP4 registers
    0000          MOVP_DEF                        ; Bit fields in MOVP registers
    0000
    0000          PACK_DEF                        ; Stack usage by exception handler
    0000          STACK_DEF                       ; Stack usage of original exception
    0000
    0000          $PSLDEF                         ; Define bit fields in PSL
    0000
    0000          .DISABLE        SUPPRESSION     ; Turn on symbol table again
    0000          .CROSS                          ; Cross reference is OK now
    0000
    0000  ; External declarations
    0000
    0000          .DISABLE        GLOBAL
    0000
    0000          .EXTERNAL -
    0000                          VAX$REFLECT_FAULT,-
    0000                          VAX$ROPRAND
    0000
    0000  ; PSECT Declarations:
    0000
    0000          .DEFAULT        DISPLACEMENT , WORD
    0000
00000000          .PSECT _VAX$CODE PIC, USR, CON, REL, LCL, SHR, EXE, RD, NOWRT, LONG
    0000
    0000          BEGIN_MARK_POINT
```

```
                              0000          .SUBTITLE      Conversion Tables
                              0000
                              0000    ;+
                              0000    ; The following tables are designed to perform fast conversions between
                              0000    ; numbers in the range 0 to 99 and their decimal equivalents. The tables
                              0000    ; are used by placing the input parameter into a register and then using
                              0000    ; the contents of that register as an index into the table.
                              0000    ;-
                              0000
                              0000    ;+
                              0000    ;       Decimal Digits to Binary Number
                              0000    ;
                              0000    ; The following table is used to convert a packed decimal byte to its binary
                              0000    ; equivalent.
                              0000    ;
                              0000    ; Packed decimal numbers that contain illegal digits in the low nibble
                              0000    ; convert as if the low nibble contained a zero. That is, the binary number
                              0000    ; will be a multiple of ten. This is done so that this table can be used to
                              0000    ; convert the least significant (highest addressed) byte of a decimal string
                              0000    ; without first masking off the sign "digit".
                              0000    ;
                              0000    ; Illegal digits in the high nibble produce UNPREDICTABLE results because the
                              0000    ; table does not contain entries to handle these illegal constructs.
                              0000    ;-
                              0000
                              0000    ;                 Binary Equivalent               Decimal Digits
                              0000    ;                 -----------------               --------------
                              0000
                              0000    DECIMAL$PACKED_TO_BINARY_TABLE::
                              0000
09 08 07 06 05 04 03 02 01 00 0000          .BYTE   00 , 01 , 02 , 03 , 04 , -      ; Index  ^X00
                              000A                  05 , 06 , 07 , 08 , 09          ;    to  ^X09
                              000A
            00 00 00 00 00 00 000A          .BYTE   00 , 00 , 00 , 00 , 00 , 00     ; Illegal decimal digits
                              0010
13 12 11 10 0F 0E 0D 0C 0B 0A 0010          .BYTE   10 , 11 , 12 , 13 , 14 , -      ; Index  ^X10
                              001A                  15 , 16 , 17 , 18 , 19          ;    to  ^X19
                              001A
            0A 0A 0A 0A 0A 0A 001A          .BYTE   10 , 10 , 10 , 10 , 10 , 10     ; Illegal decimal digits
                              0020
1D 1C 1B 1A 19 18 17 16 15 14 0020          .BYTE   20 , 21 , 22 , 23 , 24 , -      ; Index  ^X20
                              002A                  25 , 26 , 27 , 28 , 29          ;    to  ^X29
                              002A
            14 14 14 14 14 14 002A          .BYTE   20 , 20 , 20 , 20 , 20 , 20     ; Illegal decimal digits
                              0030
27 26 25 24 23 22 21 20 1F 1E 0030          .BYTE   30 , 31 , 32 , 33 , 34 , -      ; Index  ^X30
                              003A                  35 , 36 , 37 , 38 , 39          ;    to  ^X39
                              003A
            1E 1E 1E 1E 1E 1E 003A          .BYTE   30 , 30 , 30 , 30 , 30 , 30     ; Illegal decimal digits
                              0040
31 30 2F 2E 2D 2C 2B 2A 29 28 0040          .BYTE   40 , 41 , 42 , 43 , 44 , -      ; Index  ^X40
                              004A                  45 , 46 , 47 , 48 , 49          ;    to  ^X49
                              004A
            28 28 28 28 28 28 004A          .BYTE   40 , 40 , 40 , 40 , 40 , 40     ; Illegal decimal digits
                              0050
3B 3A 39 38 37 36 35 34 33 32 0050          .BYTE   50 , 51 , 52 , 53 , 54 , -      ; Index  ^X50
                              005A                  55 , 56 , 57 , 58 , 59          ;    to  ^X59
```

```
                              005A
            32 32 32 32 32 32 005A          .BYTE   50 , 50 , 50 , 50 , 50 , 50     ; Illegal decimal digits
                              0060
    45 44 43 42 41 40 3F 3E 3D 3C 0060          .BYTE   60 , 61 , 62 , 63 , 64 , -      ; Index  ^X60
                              006A                  65 , 66 , 67 , 68 , 69          ;    to  ^X69
                              006A
            3C 3C 3C 3C 3C 3C 006A          .BYTE   60 , 60 , 60 , 60 , 60 , 60     ; Illegal decimal digits
                              0070
    4F 4E 4D 4C 4B 4A 49 48 47 46 0070          .BYTE   70 , 71 , 72 , 73 , 74 , -      ; Index  ^X70
                              007A                  75 , 76 , 77 , 78 , 79          ;    to  ^X79
                              007A
            46 46 46 46 46 46 007A          .BYTE   70 , 70 , 70 , 70 , 70 , 70     ; Illegal decimal digits
                              0080
    59 58 57 56 55 54 53 52 51 50 0080          .BYTE   80 , 81 , 82 , 83 , 84 , -      ; Index  ^X80
                              008A                  85 , 86 , 87 , 88 , 89          ;    to  ^X89
                              008A
            50 50 50 50 50 50 008A          .BYTE   80 , 80 , 80 , 80 , 80 , 80     ; Illegal decimal digits
                              0090
    63 62 61 60 5F 5E 5D 5C 5B 5A 0090          .BYTE   90 , 91 , 92 , 93 , 94 , -      ; Index  ^X90
                              009A                  95 , 96 , 97 , 98 , 99          ;    to  ^X99
                              009A
            5A 5A 5A 5A 5A 5A 009A          .BYTE   90 , 90 , 90 , 90 , 90 , 90     ; Illegal decimal digits
                              00A0
                              00A0  ;+
                              00A0  ;               Binary Number    Decimal Equivalent
                              00A0  ;
                              00A0  ; The following table is used to do a fast conversion from a binary number
                              00A0  ; stored in a byte to its decimal representation. The table structure assumes
                              00A0  ; that the number lies in the range 0 to 99. Numbers that lie outside this
                              00A0  ; range produce UNPREDICTABLE results.
                              00A0  ;-
                              00A0
                              00A0  ;               Decimal Equivalents                     Binary
                              00A0  ;               -------------------                     ------
                              00A0
                              00A0  DECIMAL$BINARY_TO_PACKED_TABLE::
                              00A0
    09 08 07 06 05 04 03 02 01 00 00A0          .BYTE   ^X00 , ^X01 , ^X02 , ^X03 , ^X04 , -    ;  0 through  9
                              00AA                  ^X05 , ^X06 , ^X07 , ^X08 , ^X09        ;
                              00AA
    19 18 17 16 15 14 13 12 11 10 00AA          .BYTE   ^X10 , ^X11 , ^X12 , ^X13 , ^X14 , -    ; 10 through 19
                              00B4                  ^X15 , ^X16 , ^X17 , ^X18 , ^X19        ;
                              00B4
    29 28 27 26 25 24 23 22 21 20 00B4          .BYTE   ^X20 , ^X21 , ^X22 , ^X23 , ^X24 , -    ; 20 through 29
                              00BE                  ^X25 , ^X26 , ^X27 , ^X28 , ^X29        ;
                              00BE
    39 38 37 36 35 34 33 32 31 30 00BE          .BYTE   ^X30 , ^X31 , ^X32 , ^X33 , ^X34 , -    ; 30 through 39
                              00C8                  ^X35 , ^X36 , ^X37 , ^X38 , ^X39        ;
                              00C8
    49 48 47 46 45 44 43 42 41 40 00C8          .BYTE   ^X40 , ^X41 , ^X42 , ^X43 , ^X44 , -    ; 40 through 49
                              00D2                  ^X45 , ^X46 , ^X47 , ^X48 , ^X49        ;
                              00D2
    59 58 57 56 55 54 53 52 51 50 00D2          .BYTE   ^X50 , ^X51 , ^X52 , ^X53 , ^X54 , -    ; 50 through 59
                              00DC                  ^X55 , ^X56 , ^X57 , ^X58 , ^X59        ;
                              00DC
    69 68 67 66 65 64 63 62 61 60 00DC          .BYTE   ^X60 , ^X61 , ^X62 , ^X63 , ^X64 , -    ; 60 through 69
                              00E6                  ^X65 , ^X66 , ^X67 , ^X68 , ^X69        ;
```

```
                              00E6
79 78 77 76 75 74 73 72 71 70 00E6        .BYTE   ^X70 , ^X71 , ^X72 , ^X73 , ^X74 , -    ; 70 through 79
                              00F0                ^X75 , ^X76 , ^X77 , ^X78 , ^X79        ;
                              00F0
89 88 87 86 85 84 83 82 81 80 00F0        .BYTE   ^X80 , ^X81 , ^X82 , ^X83 , ^X84 , -    ; 80 through 89
                              00FA                ^X85 , ^X86 , ^X87 , ^X88 , ^X89        ;
                              00FA
99 98 97 96 95 94 93 92 91 90 00FA        .BYTE   ^X90 , ^X91 , ^X92 , ^X93 , ^X94 , -    ; 90 through 99
                              0104                ^X95 , ^X96 , ^X97 , ^X98 , ^X99        ;
                              0104
```

```
0104            .SUBTITLE       VAX$CMPPx - Compare Packed
0104    ;+
0104    ; Functional Description:
0104    ;
0104    ;       In 3 operand format, the source 1 string specified  by  the  length  and
0104    ;       source  1  address operands is compared to the source 2 string specified
0104    ;       by the length and source 2 address operands.  The  only  action  is  to
0104    ;       affect the condition codes.
0104    ;
0104    ;       In 4 operand format, the source 1  string  specified  by  the  source  1
0104    ;       length  and source 1 address operands is compared to the source 2 string
0104    ;       specified by the source 2 length and source  2  address  operands.  The
0104    ;       only action is to affect the condition codes.
0104    ;
0104    ; Input Parameters:
0104    ;
0104    ;       Entry at VAX$CMPP3
0104    ;
0104    ;               R0 - len.rw             Length of either decimal string
0104    ;               R1 - src1addr.ab        Address of first packed decimal string
0104    ;               R3 - src2addr.ab        Address of second packed decimal string
0104    ;
0104    ;       Entry at VAX$CMPP4
0104    ;
0104    ;               R0 - src1len.rw         Length of first packed decimal string
0104    ;               R1 - src1addr.ab        Address of first packed decimal string
0104    ;               R2 - src2len.rw         Length of second packed decimal string
0104    ;               R3 - src2addr.ab        Address of second packed decimal string
0104    ;
0104    ; Output Parameters:
0104    ;
0104    ;       R0 = 0
0104    ;       R1 = Address of the byte containing the most significant digit of
0104    ;            the first source string
0104    ;       R2 = 0
0104    ;       R3 = Address of the byte containing the most significant digit of
0104    ;            the second source string
0104    ;
0104    ; Condition Codes:
0104    ;
0104    ;       N <- first source string LSS second source string
0104    ;       Z <- first source string EQL second source string
0104    ;       V <- 0
0104    ;       C <- 0
0104    ;
0104    ; Register Usage:
0104    ;
0104    ;       This routine uses R0 through R5. The condition codes are recorded
0104    ;       in R2 as the routine executes.
0104    ;
0104    ; Algorithm:
0104    ;
0104    ;       TBS
0104    ;-
0104
0104            .SUBTITLE       Data Declarations for CMPP3 and CMPP4
0104
```

```
                          0104  ;+
                          0104  ; Define some bit fields that allow recording the presence of minus signs
                          0104  ; in either or both of the source strings.
                          0104  ;-
                          0104
                          0104          $DEFINI CMPPx_FLAGS
                          0000
                          0000          _VIELD  CMPPx,0,<-
                          0000                  <SRC1_MINUS,,M>,-
                          0000                  <SRC2_MINUS,,M>,-
                          0000                  >
                          0000
                          0000          $DEFEND CMPPx_FLAGS
                          0104
                          0104          .ENABLE         LOCAL_BLOCK
                          0104
                          0104  VAX$CMPP3::
               52  50  3C 0104          MOVZWL  R0,R2                   ; Make two source lengths equal
                   0B  11 0107          BRB     10$                     ; Only make one length check
                          0109
                          0109  VAX$CMPP4::
                          0109          ROPRAND_CHECK   R2                      ; Insure that R2 LEQU 31
                          0114  10$:    ROPRAND_CHECK   R0                      ; Insure that R0 LEQU 31
             043F  8F  BB 011C          PUSHR   #^M<R0,R1,R2,R3,R4,R5,R10>      ; Save some registers
                          0120          ESTABLISH_HANDLER       -               ; Store address of access
                          0120                  DECIMAL_ACCVIO                  ;  violation handler
                          0125
                          0125  ; Get sign of first input string
                          0125
                   54  D4 0125          CLRL    R4                      ; Assume both strings contain "+"
   55   50  04     01  EF 0127          EXTZV   #1,#4,R0,R5             ; Convert digit count to byte count
                          012C          MARK_POINT      CMPPx_ACCVIO
   55   6145   F0  8F  8B 012C          BICB3   #^B11110000,(R1)[R5],R5 ; R5 contains "sign" digit
                          0132          CASE    R5,TYPE=B,LIMIT=#10,<-  ; Dispatch on sign digit
                          0132                  30$,-                   ; 10 => sign is "+"
                          0132                  20$,-                   ; 11 => sign is "-"
                          0132                  30$,-                   ; 12 => sign is "+"
                          0132                  20$,-                   ; 13 => sign is "-"
                          0132                  30$,-                   ; 14 => sign is "+"
                          0132                  30$,-                   ; 15 => sign is "+"
                          0132                  >
                          0142
               54  01  C8 0142  20$:    BISL2   #CMPPx_M_SRC1_MINUS,R4  ; Remember that src1 contains "-"
                          0145
                          0145  ; Now get sign of second input string
                          0145
   55   52  04     01  EF 0145  30$:    EXTZV   #1,#4,R2,R5             ; Convert digit count to byte count
                          014A          MARK_POINT      CMPPx_ACCVIO
   55   6345   F0  8F  8B 014A          BICB3   #^B11110000,(R3)[R5],R5 ; R5 contains "sign" digit
                          0150          CASE    R5,TYPE=B,LIMIT=#10,<-  ; Dispatch on sign digit
                          0150                  50$,-                   ; 10 => sign is "+"
                          0150                  40$,-                   ; 11 => sign is "-"
                          0150                  50$,-                   ; 12 => sign is "+"
                          0150                  40$,-                   ; 13 => sign is "-"
                          0150                  50$,-                   ; 14 => sign is "+"
                          0150                  50$,-                   ; 15 => sign is "+"
                          0150                  >
```

```
                        0160
          54   02   C8  0160   40$:    BISL2   #CMPPx_M_SRC2_MINUS,R4  ; Remember that src2 contains "-"
                        0163
                        0163   ; At this point, we have determined the signs of both input strings. If the
                        0163   ; strings have different signs, then the comparison is done except for the
                        0163   ; extraordinary case of comparing a minus zero to a plus zero. If both signs
                        0163   ; are the same, then a digit-by-digit comparison is required.
                        0163
                        0163   50$:    CASE    R4,LIMIT=#0,TYPE=B,<-   ; Dispatch on combination of signs
                        0163                   60$,-                   ; Both signs are "+"
                        0163                   MINUS_ZERO_CHECK,-      ; Signs are different
                        0163                   MINUS_ZERO_CHECK,-      ; Signs are different
                        0163                   60$,-                   ; Both signs are "-"
                        0163                   >
                        016F
                        016F   ; Both strings have the same sign. If the strings have different lengths, then
                        016F   ; the excess digits in the longer string are checked for nonzero because that
                        016F   ; eliminates the need for further comparison.
                        016F
     55   52   50   C3  016F   60$:    SUBL3   R0,R2,R5                ; Get difference in lengths
               30   13  0173           BEQL    EQUAL_LENGTH            ; Strings have the same size
               15   19  0175           BLSS    SRC2_SHORTER            ; src2 is shorter than src1
                        0177
                        0177   ; This code executes when src1 is shorter than src2. That is, R0 LSSU R2.
                        0177   ; The large comment at the beginning of this module explains the need for the
                        0177   ; INCL R5 instruction when R0, the length of the shorter string, is odd.
                        0177
                        0177   SRC1_SHORTER:
            02 50   E9  0177           BLBC    R0,70$                  ; Skip adjustment if R0 is even
               55   D6  017A           INCL    R5                      ; Adjust digit difference if R0 is odd
55   55   04   01   EF  017C   70$:    EXTZV   #1,#4,R5,R5             ; Convert digit count to byte count
               22   13  0181           BEQL    EQUAL_LENGTH            ; Skip loop if no entire bytes in excess
                        0183
                        0183           MARK_POINT      CMPPx_ACCVIO
               83   95  0183   80$:    TSTB    (R3)+                   ; Test excess src2 digits for nonzero
               55   12  0185           BNEQ    SRC1_SMALLER            ; All done if nonzero. src1 LSS src2
            F9 55   F5  0187           SOBGTR  R5,80$                  ; Test for end of loop
                        018A
               19   11  018A           BRB     EQUAL_LENGTH            ; Enter loop that performs comparison
                        018C
                        018C   ; This code executes when src2 is shorter than src1. That is, R2 LSSU R0.
                        018C   ; The large comment at the beginning of this module explains the need for the
                        018C   ; INCL R5 instruction when R2, the length of the shorter string, is odd.
                        018C
                        018C   SRC2_SHORTER:
          50   52   D0  018C           MOVL    R2,R0                   ; R0 contains number of remaining digits
          55   55   CE  018F           MNEGL   R5,R5                   ; Make difference positive
            02 52   E9  0192           BLBC    R2,90$                  ; Skip adjustment if R2 is even
               55   D6  0195           INCL    R5                      ; Adjust digit difference if R2 is odd
55   55   04   01   EF  0197   90$:    EXTZV   #1,#4,R5,R5             ; Convert digit count to byte count
               07   13  019C           BEQL    EQUAL_LENGTH            ; Skip loop if no entire bytes in excess
                        019E
                        019E           MARK_POINT      CMPPx_ACCVIO
               81   95  019E   100$:   TSTB    (R1)+                   ; Test excess src1 digits for nonzero
               32   12  01A0           BNEQ    SRC2_SMALLER            ; All done if nonzero. src2 LSS src1
            F9 55   F5  01A2           SOBGTR  R5,100$                 ; Test for end of loop
                        01A5
```

```
                        01A5  ; All excess digits are zero. We must now perform a digit-by-digit comparison
                        01A5  ; of the remaining digits in the two strings. R0 contains the remaining number
                        01A5  ; of digits in either string.
                        01A5
                        01A5  EQUAL_LENGTH:
 50   50   04   01  EF  01A5          EXTZV   #1,#4,R0,R0                 ; Convert digit count to byte count
                08  13  01AA          BEQL    120$                        ; All done if no digits remain
                        01AC
                        01AC          MARK_POINT      CMPPx_ACCVIO
           83   81  91  01AC  110$:   CMPB    (R1)+,(R3)+                 ; Compare next two digits
                21  12  01AF          BNEQ    NOT_EQUAL                   ; Comparison complete if not equal
             F8 50  F5  01B1          SOBGTR  R0,110$                     ; Test for end of loop
                        01B4
                        01B4  ; Compare least significant digit in source and destination strings
                        01B4
                        01B4          MARK_POINT      CMPPx_ACCVIO
      51   61   0F  8B  01B4  120$:   BICB3   #^B00001111,(R1),R1         ; Strip sign from last src1 digit
                        01B8          MARK_POINT      CMPPx_ACCVIO
      53   63   0F  8B  01B8          BICB3   #^B00001111,(R3),R3         ; Strip sign from last src2 digit
           53   51  91  01BC          CMPB    R1,R3                       ; Compare least significant digits
                11  12  01BF          BNEQ    NOT_EQUAL
                        01C1
                        01C1  ; At this point, all tests have been exhausted and the two strings have
                        01C1  ; been shown to be equal. Set the Z-bit, clear the remaining condition
                        01C1  ; codes, and restore saved registers.
                        01C1
                        01C1  SRC1_EQL_SRC2:
           52   04  9A  01C1          MOVZBL  #PSL$M_Z,R2                 ; Set condition codes for src1 EQL src2
                        01C4
                        01C4  ; This is the common exit path. R2 contains the appropriate settings for the
                        01C4  ; N- and Z-bits. There is no other expected input at this point.
                        01C4
                        01C4  CMPPx_EXIT:
                6E  D4  01C4          CLRL    (SP)                        ; Set saved R0 to 0
             08 AE  D4  01C6          CLRL    8(SP)                       ; Set saved R2 to 0
                0F  B9  01C9          BICPSW  #<PSL$M_N!PSL$M_Z!PSL$M_V!PSL$M_C>      ; Start with clean slate
                52  B8  01CB          BISPSW  R2                          ; Set N- and Z-bits as appropriate
           043F 8F  BA  01CD          POPR    #^M<R0,R1,R2,R3,R4,R5,R10>      ; Restore saved registers
                    05  01D1          RSB                                     ; Return
                        01D2
                        01D2  ; The following code executes if specific digits in the two strings have
                        01D2  ; tested not equal. Separate pieces of code are selected for the two
                        01D2  ; different cases of not equal. Note that unsigned comparisons are required
                        01D2  ; here because the decimal digits "8" and "9", when appearing in the high
                        01D2  ; nibble, can cause the sign bit to be set.
                        01D2
                        01D2  NOT_EQUAL:
                08  1F  01D2          BLSSU   SRC1_SMALLER                ; Branch if src1 is smaller than src2
                        01D4
                        01D4  ; The src2 string has a smaller magnitude than the src1 string. The setting
                        01D4  ; of the signs determines how this transforms to a signed comparison. That is,
                        01D4  ; if both input signs are minus, then reverse the sense of the comparison.
                        01D4
                        01D4  SRC2_SMALLER:
        08 54   01  E0  01D4          BBS     #CMPPx_V_SRC2_MINUS,R4,SRC1_SMALLER_REALLY
                        01D8
                        01D8  ; The SRC2 string has been determined to be smaller that the SRC1 string
```

```
                          01D8
                          01D8   SRC2_SMALLER_REALLY:
              52   D4     01D8           CLRL    R2                      ; Clear both N- and Z-bits
              E8   11     01DA           BRB     CMPPx_EXIT              ; Join common exit code
                          01DC
                          01DC   ; The src1 string has a smaller magnitude than the src2 string. The setting
                          01DC   ; of the signs determines how this transforms to a signed comparison. That is,
                          01DC   ; if both input signs are minus, then reverse the sense o: the comparison.
                          01DC
                          01DC   SRC1_SMALLER:
       F8 54   00   E0    01DC           BBS     #CMPPx_V_SRC1_MINUS,R4,SRC2_SMALLER_REALLY
                          01E0
                          01E0   ; The src1 string has been determined to be smaller that the src2 string
                          01E0
                          01E0   SRC1_SMALLER_REALLY:
          52   08   90    01E0           MOVB    #PSL$M_N,R2             ; Clear both N- and Z-bits
               DF   11    01E3           BRB     CMPPx_EXIT              ; Join common exit code
                          01E5
                          01E5   ; The following code executes if the two input strings have different
                          01E5   ; signs. We need to determine if a comparison between plus zero and minus
                          01E5   ; zero is being made, because such a comparison should test equal. We scan
                          01E5   ; first one string and then the other. If we find a nonzero digit anywhere
                          01E5   ; along the way, we immediately exit this test and set the final condition
                          01E5   ; codes such that the "-" string is smaller than the "+" string. If we
                          01E5   ; exhaust both strings without finding a nonzero digit, then we report
                          01E5   ; that the two strings are equal.
                          01E5
                          01E5   MINUS_ZERO_CHECK:
  50   50   04   01   EF  01E5           EXTZV   #1,#4,R0,R0             ; Convert src1 digit count to byte count
                 07   13  01EA           BEQL    170$                    ; Skip loop if only single digit
                          01EC
                          01EC           MARK_POINT      CMPPx_ACCVIO
                 81   95  01EC   160$:   TSTB    (R1)+                   ; Test next byte for nonzero
                 1D   12  01EE           BNEQ    CMPPx_NOT_ZERO          ; Exit loop if nonzero
              F9 50   F5  01F0           SOBGTR  R0,160$                 ; Test for end of loop
                          01F3
                          01F3           MARK_POINT      CMPPx_ACCVIO
      61     F0 8F   93   01F3   170$:   BITB    #^B11110000,(R1)        ; Test least significant digit
                 14   12  01F7           BNEQ    CMPPx_NOT_ZERO          ; Exit if this digit is not zero
                          01F9
                          01F9   ; All digits in src1 are zero. Now we must look for nonzero digits in src2.
                          01F9
  52   52   04   01   EF  01F9           EXTZV   #1,#4,R2,R2             ; Convert src2 digit count to byte count
                 07   13  01FE           BEQL    190$                    ; Skip loop if only single digit
                          0200
                          0200           MARK_POINT      CMPPx_ACCVIO
                 83   95  0200   180$:   TSTB    (R3)+                   ; Test next byte for nonzero
                 09   12  0202           BNEQ    CMPPx_NOT_ZERO          ; Exit loop if nonzero
              F9 52   F5  0204           SOBGTR  R2,180$                 ; Test for end of loop
                          0207
                          0207           MARK_POINT      CMPPx_ACCVIO
      63     F0 8F   93   0207   190$:   BITB    #^B11110000,(R3)        ; Test least significant digit
                 B4   13  020B           BEQL    SRC1_EQL_SRC2           ; Branch if two strings are equal
                          020D
                          020D   ; At least one of the two input strings contains at least one nonzero digit.
                          020D   ; That knowledge is sufficient to determine the result of the comparison
                          020D   ; based simply on the two (necessarily different) signs of the input strings.
```

```
                       020D
                       020D    CMPPx_NOT_ZERO:
C7 54   01   E0        020D            BBS     #CMPPx_V_SRC2_MINUS,R4,SRC2_SMALLER_REALLY
        CD   11        0211            BRB     SRC1_SMALLER_REALLY
                       0213
                       0213            .DISABLE        LOCAL_BLOCK
```

VAX
Pse

PSE

$AB
 VA
PC
HAR

Pha

Ini
Com
Pas
Sym
Pas
Sym
Pse
Cro
Ass

The
245
The
115
22

Mac

_$2
_$2
TOT

272

The

MAC

```
                       0213          .SUBTITLE       VAX$MOVP - Move Packed
                       0213     ;+
                       0213     ; Functional Description:
                       0213     ;
                       0213     ;       The destination string specified by the length and  destination  address
                       0213     ;       operands  is  replaced  by the source string specified by the length and
                       0213     ;       source address operands.
                       0213     ;
                       0213     ; Input Parameters:
                       0213     ;
                       0213     ;       R0 - len.rw             Length of input and output decimal strings
                       0213     ;       R1 - srcaddr.ab         Address of input packed decimal string
                       0213     ;       R3 - dstaddr.ab         Address of output packed decimal string
                       0213     ;
                       0213     ;       PSL<C>                  Contains setting of C-bit when MOVP executed
                       0213     ;
                       0213     ; Output Parameters:
                       0213     ;
                       0213     ;       R0 = 0
                       0213     ;       R1 = Address of byte containing most significant digit of
                       0213     ;            the source string
                       0213     ;       R2 = 0
                       0213     ;       R3 = Address of byte containing most significant digit of
                       0213     ;            the destination string
                       0213     ;
                       0213     ; Condition Codes:
                       0213     ;
                       0213     ;       N <- destination string LSS 0
                       0213     ;       Z <- destination string EQL 0
                       0213     ;       V <- 0
                       0213     ;       C <- C                          ; Note that C-bit is preserved!
                       0213     ;
                       0213     ; Register Usage:
                       0213     ;
                       0213     ;       This routine uses R0 through R3. The condition codes are recorded
                       0213     ;       in R2 as the routine executes.
                       0213     ;
                       0213     ; Notes:
                       0213     ;
                       0213     ;       The initial value of the C-bit must be captured (saved in R2) before
                       0213     ;       any instructions execute that alter the C-bit.
                       0213     ;-
                       0213
                       0213     VAX$MOVP::
           52    DC    0213          MOVPSL  R2                      ; Save initial PSL (to preserve C-bit)
                       0215
                       0215          ASSUME MOVP_B_STATE EQ 2        ; Make sure that FPD bit is in R0<23:16>
                       0215
  05 50    14    E1    0215          BBC     #<MOVP_V_FPD + 16>,R0,5$        ; Branch if first time
           10    EF    0219          EXTZV   #<MOVP_V_SAVED_PSW + 16>,-      ; Otherwise, replace condition
52    50   04          021B                  #MOVP_S_SAVED_PSW,R0,R2         ;  codes with previous settings
                       021E
                       021E     5$:    ROPRAND_CHECK   R0              ; Insure that R0 LEQU 31
                       0229
                       0229     ; Save the starting addresses of the input and output strings in addition to
                       0229     ; the digit count operand (initial R0 contents.) Store a place holder for
                       0229     ; saved R2.
```

```
                         0229
           040F 8F   BB  0229               PUSHR   #^M<R0,R1,R2,R3,R10>    ; Save initial register contents
                         022D               ESTABLISH_HANDLER       -       ; Store address of access
                         022D                       DECIMAL_ACCVIO          ;  violation handler
                         0232
 52   03   01   02   F0  0232               INSV    #<PSL$M_Z@-1>,#1,#3,R2  ; Set Z-bit. Clear N- and V-bits.
 50   50   04   01   EF  0237               EXTZV   #1,#4,R0,R0             ; Convert digit count to byte count
                0B   13  023C               BEQL    30$                     ; Skip loop if zero or one digit
                         023E
                         023E               MARK_POINT      MOVP_ACCVIO
           83   81   90  023E       10$:    MOVB    (R1)+,(R3)+             ; Move next two digits
                03   13  0241               BEQL    20$                     ; Leave Z-bit alone if both zero
           52   04   8A  0243               BICB    #PSL$M_Z,R2             ; Otherwise, clear saved Z-bit
           F5   50   F5  0246       20$:    SOBGTR  R0,10$                  ; Check for end of loop
                         0249
                         0249       ; The last byte must be processed in a special way. The digit must be checked
                         0249       ; for nonzero because that affects the condition codes. The sign must be
                         0249       ; transformed into the preferred form. The N-bit must be set if the input
                         0249       ; is negative, but cleared in the case of negative zero.
                         0249
                         0249               MARK_POINT      MOVP_ACCVIO
           50   61   90  0249       30$:    MOVB    (R1),R0                 ; Get last input byte (R1 now scratch)
      50   F0   8F   93  024C               BITB    #^B11110000,R0          ; Is digit nonzero?
                03   13  0250               BEQL    40$                     ; Branch if zero
           52   04   8A  0252               BICB    #PSL$M_Z,R2             ; Otherwise, clear saved Z-bit
 51   50   F0   8F   8B  0255       40$:    BICB3   #^B11110000,R0,R1       ; Sign "digit" to R1
                         025A
                         025A       ; Assume that the sign is "+". If the input sign is minus, one of the several
                         025A       ; fixups that must be done is to change the output sign from "+" to "-".
                         025A
 50   04   00   0C   F0  025A               INSV    #12,#0,#4,R0            ; 12 is preferred plus sign
                         025F               CASE    R1,LIMIT=#10,TYPE=B,<-  ; Dispatch on sign type
                         025F                       60$,-                   ; 10 => +
                         025F                       50$,-                   ; 11 => -
                         025F                       60$,-                   ; 12 => +
                         025F                       50$,-                   ; 13 => -
                         025F                       60$,-                   ; 14 => +
                         025F                       60$,-                   ; 15 => +
                         025F                       >
                         026F
                         026F       ; Input sign is "-"
                         026F
      05   52   02   E0  026F       50$:    BBS     #PSL$V_Z,R2,60$         ; Treat as "+" if negative zero
                50   D6  0273               INCL    R0                      ; 13 is preferred minus sign
           52   08   88  0275               BISB    #PSL$M_N,R2             ; Set N-bit
                         0278
                         0278       ; Input sign is "+" or input is negative zero. Nothing special to do.
                         0278
                         0278               MARK_POINT      MOVP_ACCVIO
           63   50   90  0278       60$:    MOVB    R0,(R3)                 ; Move modified final digit
                6E   D4  027B               CLRL    (SP)                    ; R0 and R2 must be zero on output
           08   AE   D4  027D               CLRL    8(SP)                   ;  so clear saved R0 and R2
                0F   B9  0280               BICPSW  #<PSL$M_N!PSL$M_Z!PSL$M_V!PSL$M_C>      ; Clear all codes
                52   B8  0282               BISPSW  R2                      ; Reset codes as appropriate
           040F 8F   BA  0284               POPR    #^M<R0,R1,R2,R3,R10>    ; Restore saved registers
                     05  0288               RSB                             ; Return
```

```
0289        .SUBTITLE      Routine to Strip Leading Zeros from Decimal String
0289  ;+
0289  ; Functional Description:
0289  ;
0289  ;       This routine strips leading (high-order) zeros from a packed decimal
0289  ;       string. The routine exists based on two assumptions.
0289  ;
0289  ;       1.  Many of the decimal strings that are used in packed decimal
0289  ;           operations have several leading zeros.
0289  ;
0289  ;       2.  The operations that are performed on a byte containing packed
0289  ;           decimal digits are more complicated that the combination of this
0289  ;           routine and any special end processing that occurs in the various
0289  ;           VAX$xxxxxx routines when a string is exhausted.
0289  ;
0289  ;       This routine exists as a performance enhancement. As such, it can only
0289  ;       succeed if it is extremely efficient. It does not attempt to be
0289  ;       rigorous in squeezing every last zero out of a string. It eliminates
0289  ;       only entire bytes that contain two zero digits. It does not look for a
0289  ;       leading zero in the high order nibble of a string of odd length.
0289  ;
0289  ;       The routine also assumes that the input decimal strings are well
0289  ;       formed. If an even-length decimal string does not have a zero in its
0289  ;       unused high order nibble, then no stripping takes place, even though
0289  ;       the underlying VAX$xxxxxx routine will work correctly.
0289  ;
0289  ;       (The comment in the next four lines is preserved for its historical
0289  ;       content.)
0289  ;
0289  ;       Finally, there is no explicit test for the end of the string. The
0289  ;       routine assumes that the low order byte, the one that contains the
0289  ;       sign, is not equal to zero. This can cause rather strange behavior
0289  ;       (read UNPREDICTABLE) for poorly formed decimal strings.
0289  ;
0289  ;       (The following comment describes the revised treatment of certain forms
0289  ;       of illegal packed decimal strings.)
0289  ;
0289  ;       Although an end-of-string test is not required for well formed packed
0289  ;       decimal strings, it turns out that some layered products create packed
0289  ;       decimal data on the fly consisting of so many bytes containing zero. In
0289  ;       other words, the sign nibble contains zero. Previous implementations of
0289  ;       the VAX architecture have treated these strings as representations of
0289  ;       packed decimal zero.
0289  ;
0289  ;       The BLEQ 30$ instructions that exist in the following two loops detect
0289  ;       these strings and treat them as strings with a digit count of one.
0289  ;       (The digit itself is zero.) Whether this string is treated as +0 or -0
0289  ;       is determined by the caller of this routine. That much UNPREDICTABLE
0289  ;       behavior remains in the treatment of these illegal strings.
0289  ;
0289  ;       (End of revised comment)
0289  ;
0289  ; Input and Output Parameters:
0289  ;
0289  ;       There are really two identical but separate routines here. One is
0289  ;       used when the input decimal string descriptor is in R0 and R1. The
0289  ;       other is used when R2 and R3 describe the decimal string. Note that
```

```
                  0289  ;       we have already performed the reserved operand checks so that R0 (or
                  0289  ;       R2) is guaranteed LEQU 31.
                  0289  ;
                  0289  ;       If the high order digit of an initially even length string is zero,
                  0289  ;       then the digit count (R0 or R2) is reduced by one. For all other
                  0289  ;       cases, the digit count is reduced by two as an entire byte of zeros
                  0289  ;       is skipped.
                  0289  ;
                  0289  ; Input Parameters (for entry at DECIMAL$STRIP_ZEROS_R0_R1):
                  0289  ;
                  0289  ;       R0<4:0> - len.rw        Length of input decimal string
                  0289  ;       R1      - addr.ab       Address of input packed decimal string
                  0289  ;
                  0289  ; Output Parameters (for entry at DECIMAL$STRIP_ZEROS_R0_R1):
                  0289  ;
                  0289  ;       R1      Advanced to first nonzero byte in string
                  0289  ;       R0      Reduced accordingly (Note that if R0 is altered at all,
                  0289  ;               then R0 is always ODD on exit.)
                  0289  ;
                  0289  ; Input Parameters (for entry at DECIMAL$STRIP_ZEROS_R2_R3):
                  0289  ;
                  0289  ;       R2<4:0> - len.rw        Length of input decimal string
                  0289  ;       R3      - addr.ab       Address of input packed decimal string
                  0289  ;
                  0289  ; Output Parameters (for entry at DECIMAL$STRIP_ZEROS_R2_R3):
                  0289  ;
                  0289  ;       R3      Advanced to first nonzero byte in string
                  0289  ;       R2      Reduced accordingly (Note that if R2 is altered at all,
                  0289  ;               then R2 is always ODD on exit.)
                  0289  ;
                  0289  ; Note:
                  0289  ;
                  0289  ;       Although these routines can generate access violations, there is no
                  0289  ;       MARK_POINT here because these routines can be called from other
                  0289  ;       modules (and are not called by the routines in this module). The PC
                  0289  ;       check is made based on the return PC from this subroutine rather than
                  0289  ;       on the PC of the instruction that accessed the inaccessible address.
                  0289  ;-
                  0289
                  0289  ; This routine is used when the decimal string is described by R0 (digit
                  0289  ; count) and R1 (string address).
                  0289
                  0289  DECIMAL$STRIP_ZEROS_R0_R1::
   06 50    E8    0289          BLBS    R0,10$                  ; Skip first check if R0 starts out ODD
      81    95    028C          TSTB    (R1)+                   ; Is first byte zero?
      0D    12    028E          BNEQ    20$                     ; All done if not
      50    D7    0290          DECL    R0                      ; Skip leading zero digit (R0 NEQU 0)
                  0292
      81    95    0292  10$:    TSTB    (R1)+                   ; Is next byte zero?
      07    12    0294          BNEQ    20$                     ; All done if not
 50   02    C2    0296          SUBL    #2,R0                   ; Decrease digit count by 2
      05    15    0299          BLEQ    30$                     ; We passed the end of the string
      F5    11    029B          BRB     10$                     ; ... and charge on
                  029D
      51    D7    029D  20$:    DECL    R1                      ; Back up R1 to last nonzero byte
            05    029F          RSB
                  02A0
```

```
      50 02   C0 02A0   30$:    ADDL    #2,R0                   ; Undo last R0 modification
         F8   11 02A3           BRB     20$                     ; ... and take common exit
                 02A5
                 02A5   ; This routine is used when the decimal string is described by R2 (digit
                 02A5   ; count) and R3 (string address).
                 02A5
                 02A5   DECIMAL$STRIP_ZEROS_R2_R3::
   06 52      E8 02A5           BLBS    R2,10$                  ; Skip first check if R2 starts out ODD
         83   95 02A8           TSTB    (R3)+                   ; Is first byte zero?
         0D   12 02AA           BNEQ    20$                     ; All done if not
         52   D7 02AC           DECL    R2                      ; Skip leading zero digit (R2 NEQU 0)
                 02AE
         83   95 02AE   10$:    TSTB    (R3)+                   ; Is next byte zero?
         07   12 02B0           BNEQ    20$                     ; All done if not
      52 02   C2 02B2           SUBL    #2,R2                   ; Decrease digit count by 2
         05   15 02B5           BLEQ    30$                     ; We passed the end of the string
         F5   11 02B7           BRB     10$                     ; ... and charge on
                 02B9
         53   D7 02B9   20$:    DECL    R3                      ; Back up R3 to last nonzero byte
              05 02BB           RSB
                 02BC
      52 02   C0 02BC   30$:    ADDL    #2,R2                   ; Undo last R2 modification
         F8   11 02BF           BRB     20$                     ; ... and take common exit
                 02C1
```

```
                02C1          .SUBTITLE       DECIMAL_ROPRAND
                02C1   ;-
                02C1   ; Functional Description:
                02C1   ;
                02C1   ;       This routine receives control when a digit count larger than 31
                02C1   ;       is detected. The exception is architecturally defined as an
                02C1   ;       abort so there is no need to store intermediate state. Because
                02C1   ;       all of the routines in this module check for legal digit counts
                02C1   ;       before saving any registers, this routine simply passes control
                02C1   ;       to VAX$ROPRAND.
                02C1   ;
                02C1   ; Input Parameters:
                02C1   ;
                02C1   ;       0(SP) - Return PC from VAX$xxxxxx routine
                02C1   ;
                02C1   ; Output Parameters:
                02C1   ;
                02C1   ;       0(SP) - Offset in packed register array to delta PC byte
                02C1   ;       4(SP) - Return PC from VAX$xxxxxx routine
                02C1   ;
                02C1   ; Implicit Output:
                02C1   ;
                02C1   ;       This routine passes control to VAX$ROPRAND where further
                02C1   ;       exception processing takes place.
                02C1   ;-
                02C1
                02C1          ASSUME CMPP3_B_DELTA_PC EQ MOVP_B_DELTA_PC
                02C1          ASSUME CMPP4_B_DELTA_PC EQ MOVP_B_DELTA_PC
                02C1
                02C1   DECIMAL_ROPRAND:
      03    DD  02C1          PUSHL   #MOVP_B_DELTA_PC        ; Store offset to delta PC byte
   FD3A'    31  02C3          BRW     VAX$ROPRAND             ; Pass control along
```

```
                          02C6         .SUBTITLE      DECIMAL_ACCVIO - Reflect an Access Violation
                          02C6   ;+
                          02C6   ; Functional Description:
                          02C6   ;
                          02C6   ;       This routine receives control when an access violation occurs while
                          02C6   ;       executing within the emulator routines for CMPP3, CMPP4 or MOVP.
                          02C6   ;
                          02C6   ;       The routine header for ASHP_ACCVIO in module VAX$ASHP contains a
                          02C6   ;       detailed description of access violation handling for the decimal
                          02C6   ;       string instructions.
                          02C6   ;
                          02C6   ; Input Parameters:
                          02C6   ;
                          02C6   ;       See routine ASHP_ACCVIO in module VAX$ASHP
                          02C6   ;
                          02C6   ; Output Parameters:
                          02C6   ;
                          02C6   ;       See routine ASHP_ACCVIO in module VAX$ASHP
                          02C6   ;-
                          02C6
                          02C6   DECIMAL_ACCVIO:
              52    D4    02C6         CLRL    R2                         ; Initialize the counter
         FD34 CF    9F    02C8         PUSHAB  MODULE_BASE                ; Store base address of this module
           51 8E    C2    02CC         SUBL2   (SP)+,R1                   ; Get PC relative to this base
                          02CF
   0000'CF42  51    B1    02CF   10$:  CMPW    R1,PC_TABLE_BASE[R2]       ; Is this the right PC?
              07    13    02D5         BEQL    30$                        ; Exit loop if true
      F4 52   0E    F2    02D7         AOBLSS  #TABLE_SIZE,R2,10$         ; Do the entire table
                          02DB
                          02DB   ; If we drop through the dispatching based on PC, then the exception is not
                          02DB   ; one that we want to back up. We simply reflect the exception to the user.
                          02DB
              0F    BA    02DB   20$:  POPR    #^M<R0,R1,R2,R3>           ; Restore saved registers
                    05    02DD         RSB                                ; Return to exception dispatcher
                          02DE
                          02DE   ; The exception PC matched one of the entries in our PC table. R2 contains
                          02DE   ; the index into both the PC table and the handler table. R1 has served
                          02DE   ; its purpose and can be used as a scratch register.
                          02DE
 51   0000'CF42     3C    02DE   30$:  MOVZWL  HANDLER_TABLE_BASE[R2],R1      ; Get the offset to the handler
      FD17 CF41     17    02E4         JMP     MODULE_BASE[R1]            ; Pass control to the handler
                          02E9
                          02E9   ; In all of the instruction-specific routines, the state of the stack
                          02E9   ; will be shown as it was when the exception occurred. All offsets will
                          02E9   ; be pictured relative to R0.
```

```
                        02E9          .SUBTITLE       Context-Specific Access Violation Handling for VAX$CMPPx
                        02E9    ;+
                        02E9    ; Functional Description:
                        02E9    ;
                        02E9    ;       It is trivial to back out CMPP3 and CMPP4 because neither of these
                        02E9    ;       routines uses any stack space (other than saved register space). The
                        02E9    ;       only reason that this routine does not use the common
                        02E9    ;       VAX$DECIMAL_ACCVIO exit path is that fewer registers are saved by
                        02E9    ;       these two routines than are saved by the typical packed decimal
                        02E9    ;       emulation routine.
                        02E9    ;
                        02E9    ; Input Parameters:
                        02E9    ;
                        02E9    ;       R0 - Address of top of stack when access violation occurred
                        02E9    ;
                        02E9    ;       00(R0) - Saved R0 on entry to VAX$CMPPx
                        02E9    ;       04(R0) - Saved R1
                        02E9    ;       08(R0) - Saved R2
                        02E9    ;       12(R0) - Saved R3
                        02E9    ;       16(R0) - Saved R4
                        02E9    ;       20(R0) - Saved R5
                        02E9    ;       24(R0) - Saved R10
                        02E9    ;       28(R0) - Return PC from VAX$CMPPx routine
                        02E9    ;
                        02E9    ;       00(SP) - Saved R0 (restored by VAX$HANDLER)
                        02E9    ;       04(SP) - Saved R1
                        02E9    ;       08(SP) - Saved R2
                        02E9    ;       12(SP) - Saved R3
                        02E9    ;
                        02E9    ; Output Parameters:
                        02E9    ;
                        02E9    ;       R0 is advanced over saved register array as the registers are restored.
                        02E9    ;       R0 ends up pointing at the return PC.
                        02E9    ;
                        02E9    ;       R1 contains the value of delta PC for all of the routines that
                        02E9    ;       use this common code path. The FPD and ACCVIO bits are both set
                        02E9    ;       in R1.
                        02E9    ;
                        02E9    ;       00(R0) - Return PC from VAX$CMPPx routine
                        02E9    ;
                        02E9    ;       00(SP) - Value of R0 on entry to VAX$CMPPx
                        02E9    ;       04(SP) - Value of R1 on entry to VAX$CMPPx
                        02E9    ;       08(SP) - Value of R2 on entry to VAX$CMPPx
                        02E9    ;       12(SP) - Value of R3 on entry to VAX$CMPPx
                        02E9    ;
                        02E9    ;       R4, R5, and R10 are restored to their values on entry to VAX$CMPPx.
                        02E9    ;-
                        02E9
                        02E9          .ENABLE         LOCAL_BLOCK
                        02E9
                        02E9    CMPPx_ACCVIO::
      6E     80    7D   02E9          MOVQ    (R0)+,PACK_L_SAVED_R0(SP)       ; "Restore" R0 and R1
   08 AE     80    7D   02EC          MOVQ    (R0)+,PACK_L_SAVED_R2(SP)       ; "Restore" R2 and R3
      54     80    7D   02F0          MOVQ    (R0)+,R4                        ; Really restore R4 and R5
                        02F3
                        02F3    ; The last two instructions can be shared with MOVP_ACCVIO, provided that
                        02F3    ; the following assumptions hold.
```

```
            02F3  1083
            02F3  1084          ASSUME CMPP3_B_DELTA_PC EQ MOVP_B_DELTA_PC
            02F3  1085          ASSUME CMPP4_B_DELTA_PC EQ MOVP_B_DELTA_PC
            02F3  1086
   0D   11  02F3  1087          BRB     10$                     ; Share remainder with MOVP_ACCVIO
```

```
                02F5          .SUBTITLE       Context-Specific Access Violation Handling for VAX$MOVP
                02F5  ;+
                02F5  ; Functional Description:
                02F5  ;
                02F5  ;       It is almost too trivial to back out VAX$MOVP to its starting point.
                02F5  ;       If time permits, we will add restart points to this routine. This will
                02F5  ;       illustrate how one could go about adding restart capability to other
                02F5  ;       decimal instructions, allowing the routines to pick up where they left
                02F5  ;       off if an access violation occurs. This will also point out the
                02F5  ;       magnitude of the task by showing the amount of intermediate state that
                02F5  ;       must be saved for even so simple a routine as VAX$MOVP.
                02F5  ;
                02F5  ;       The VAX$MOVP routine, like VAX$CMPPx, uses no stack space. It also
                02F5  ;       saves only a subset of the registers and so a special exit path must
                02F5  ;       be taken to VAX$REFLECT_FAULT.
                02F5  ;
                02F5  ; Input Parameters:
                02F5  ;
                02F5  ;       R0 - Address of top of stack when access violation occurred
                02F5  ;
                02F5  ;       00(R0) - Saved R0 on entry to VAX$MOVP
                02F5  ;       04(R0) - Saved R1
                02F5  ;       08(R0) - Saved R2
                02F5  ;       12(R0) - Saved R3
                02F5  ;       16(R0) - Saved R10
                02F5  ;       20(R0) - Return PC from VAX$MOVP routine
                02F5  ;
                02F5  ;       00(SP) - Saved R0 (restored by VAX$HANDLER)
                02F5  ;       04(SP) - Saved R1
                02F5  ;       08(SP) - Saved R2
                02F5  ;       12(SP) - Saved R3
                02F5  ;
                02F5  ; Output Parameters:
                02F5  ;
                02F5  ;       R0 is advanced over saved register array as the registers are restored.
                02F5  ;       R0 ends up pointing at the return PC.
                02F5  ;
                02F5  ;       R1 contains the value of delta PC for all of the routines that
                02F5  ;       use this common code path. The FPD and ACCVIO bits are both set
                02F5  ;       in R1.
                02F5  ;
                02F5  ;       00(R0) - Return PC from VAX$MOVP routine
                02F5  ;
                02F5  ;       00(SP) - Value of R0 on entry to VAX$MOVP
                02F5  ;       04(SP) - Value of R1 on entry to VAX$MOVP
                02F5  ;       08(SP) - Value of R2 on entry to VAX$MOVP
                02F5  ;       12(SP) - Value of R3 on entry to VAX$MOVP
                02F5  ;
                02F5  ;       R10 is restored to its value on entry to VAX$MOVP.
                02F5  ;-
                02F5
                02F5  MOVP_ACCVIO::
     6E    80   7D  02F5          MOVQ    (R0)+,PACK_L_SAVED_R0(SP)       ; "Restore" R0 and R1
  08 AE    80   7D  02F8          MOVQ    (R0)+,PACK_L_SAVED_R2(SP)       ; "Restore" R2 and R3
  F8 A0    10   89  02FC          BISB3   #MOVP_M_FPD,-8(R0),-
     02 AE          0300                  MOVP_B_STATE(SP)                ; Preserve saved C-bit
                    0302
```

```
                5A   80   D0  0302  1146 10$:    MOVL    (R0)+,R10                       ; Really restore R10
                              0305  1147
       51  00000303 8F   D0  0305  1148          MOVL    #<MOVP_B_DELTA_PC!-        ; Indicate offset for delta PC
                              030C  1149                  PACK_M_FPD!-               ; FPD bit should be set
                              030C  1150                  PACK_M_ACCVIO>,R1          ; This is an access violation
                 FCF1'   31  030C  1151          BRW     VAX$REFLECT_FAULT          ; Continue exception handling
                              030F  1152
                              030F  1153          .DISABLE        LOCAL_BLOCK
                              030F  1154
                              030F  1155          .END
```

```
...PC...                        = 00000278
...ROPRAND...                   = 00000223 R      02
CMPP3_B_DELTA_PC                = 00000003
CMPP4_B_DELTA_PC                = 00000003
CMPPX_ACCVIO                      000002E9 RG     02
CMPPX_EXIT                        000001C4 R      02
CMPPX_M_SRC1_MINUS              = 00000001
CMPPX_M_SRC2_MINUS              = 00000002
CMPPX_NOT_ZERO                    0000020D R      02
CMPPX_V_SRC1_MINUS              = 00000000
CMPPX_V_SRC2_MINUS              = 00000001
DECIMAL$BINARY_TO_PACKED_TABLE    000000A0 RG     02
DECIMAL$PACKED_TO_BINARY_TABLE    00000000 RG     02
DECIMAL$STRIP_ZEROS_R0_R1         00000289 RG     02
DECIMAL$STRIP_ZEROS_R2_R3         000002A5 RG     02
DECIMAL_ACCVIO                    000002C6 R      02
DECIMAL_ROPRAND                   000002C1 R      02
EQUAL_LENGTH                      000001A5 R      02
HANDLER_TABLE_BASE                00000000 R      04
MINUS_ZERO_CHECK                  000001E5 R      02
MODULE_BASE                     = 00000000 R      02
MOVP_ACCVIO                       000002F5 RG     02
MOVP_B_DELTA_PC                 = 00000003
MOVP_B_STATE                    = 00000002
MOVP_M_FPD                      = 00000010
MOVP_S_SAVED_PSW                = 00000004
MOVP_V_FPD                      = 00000004
MOVP_V_SAVED_PSW                = 00000000
NOT_EQUAL                         000001D2 R      02
PACK_L_SAVED_R0                 = 00000000
PACK_L_SAVED_R2                 = 00000008
PACK_M_ACCVIO                   = 00000200
PACK_M_FPD                      = 00000100
PC_TABLE_BASE                     00000000 R      03
PSL$M_C                         = 00000001
PSL$M_N                         = 00000008
PSL$M_V                         = 00000002
PSL$M_Z                         = 00000004
PSL$V_Z                         = 00000002
SIZ...                          = 00000001
SRC1_EQL_SRC2                     000001C1 R      02
SRC1_SHORTER                      00000177 R      02
SRC1_SMALLER                      000001DC R      02
SRC1_SMALLER_REALLY               000001E0 R      02
SRC2_SHORTER                      0000018C R      02
SRC2_SMALLER                      000001D4 R      02
SRC2_SMALLER_REALLY               000001D8 R      02
TABLE_SIZE                      = 0000000E
VAX$CMPP3                         00000104 RG     02
VAX$CMPP4                         00000109 RG     02
VAX$MOVP                          00000213 RG     02
VAX$REFLECT_FAULT                 ********    X   00
VAX$ROPRAND                       ********    X   00
```

! Psect synopsis !

PSECT name	Allocation		PSECT No.	Attributes										
. ABS .	00000000	(0.)	00 (0.)	NOPIC	USR	CON	ABS	LCL	NOSHR	NOEXE	NORD	NOWRT	NOVEC	BYTE
ABS	00000000	(0.)	01 (1.)	NOPIC	USR	CON	ABS	LCL	NOSHR	EXE	RD	WRT	NOVEC	BYTE
_VAX$CODE	0000030F	(783.)	02 (2.)	PIC	USR	CON	REL	LCL	SHR	EXE	RD	NOWRT	NOVEC	LONG
PC_TABLE	0000001C	(28.)	03 (3.)	PIC	USR	CON	REL	LCL	SHR	NOEXE	RD	NOWRT	NOVEC	BYTE
HANDLER_TABLE	0000001C	(28.)	04 (4.)	PIC	USR	CON	REL	LCL	SHR	NOEXE	RD	NOWRT	NOVEC	BYTE

! Performance indicators !

Phase	Page faults	CPU Time	Elapsed Time
Initialization	10	00:00:00.05	00:00:01.39
Command processing	70	00:00:00.47	00:00:04.45
Pass 1	139	00:00:04.42	00:00:17.55
Symbol table sort	0	00:00:00.20	00:00:00.79
Pass 2	205	00:00:02.25	00:00:06.44
Symbol table output	6	00:00:00.07	00:00:00.39
Psect synopsis output	3	00:00:00.03	00:00:00.03
Cross-reference output	0	00:00:00.00	00:00:00.00
Assembler run totals	433	00:00:07.49	00:00:31.04

The working set limit was 1200 pages.
24566 bytes (48 pages) of virtual memory were used to buffer the intermediate code.
There were 10 pages of symbol table space allocated to hold 141 non-local and 43 local symbols.
1155 source lines were read in Pass 1, producing 20 object records in Pass 2.
22 pages of virtual memory were used to define 20 macros.

! Macro library statistics !

Macro library name	Macros defined
_255DUA28:[EMULAT.OBJ]VAXMACROS.MLB;1	10
_255DUA28:[SYSLIB]STARLET.MLB;2	6
TOTALS (all libraries)	16

272 GETS were required to define 16 macros.

There were no errors, warnings or information messages.

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

0144 AH-BT13A-SE
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY

VAXHANDLR
LIS

VAXCVTLP
LIS

VAXDECIML
LIS

VAXEMULAT
LIS

VAXCVTPL
LIS

VAXEDITPC
LIS