

```
EEEEEEEEEEEEEEEEEE MMM      MMM      UUU      UUU      LLL      AAAAAAAAAA      TTTTTTTTTTTTTTTT
EEEEEEEEEEEEEEEEEE MMM      MMM      UUU      UUU      LLL      AAAAAAAAAA      TTTTTTTTTTTTTTTT
EEEEEEEEEEEEEEEEEE MMM      MMM      UUU      UUU      LLL      AAAAAAAAAA      TTTTTTTTTTTTTTTT
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      UUU      UUU      LLL      AAA      AAA      TTT
EEE      MMM      MMM      UUU      UUU      LLL      AAA      AAA      TTT
EEEEEEEEEEEEEEEEEE MMM      MMM      UUU      UUU      LLL      AAA      AAA      TTT
EEEEEEEEEEEEEEEEEE MMM      MMM      UUU      UUU      LLL      AAA      AAA      TTT
EEEEEEEEEEEEEEEEEE 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
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
EEEEEEEEEEEEEEEEEE MMM      MMM      UUUUUUUUUUUUUUUU      LLLLLLLLLLLLLLLLLL      AAA      AAA      TTT
EEEEEEEEEEEEEEEEEE MMM      MMM      UUUUUUUUUUUUUUUU      LLLLLLLLLLLLLLLLLL      AAA      AAA      TTT
EEEEEEEEEEEEEEEEEE MMM      MMM      UUUUUUUUUUUUUUUU      LLLLLLLLLLLLLLLLLL      AAA      AAA      TTT
```

[illegible]

```

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
LLLLLLLLLLLL IIIIII          SSSSSSSS
LLLLLLLLLLLL IIIIII          SSSSSSSS

```

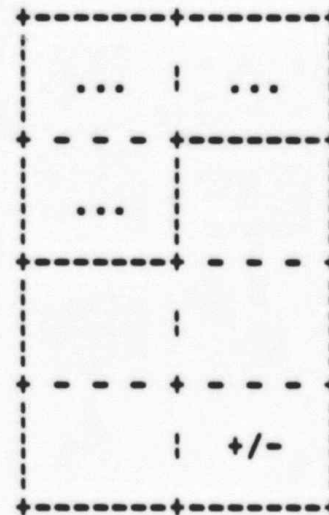

(2)	87	Miscellaneous Notes
(5)	242	Declarations
(6)	277	Conversion Tables
(7)	402	VAX\$CMPPx - Compare Packed
(7)	457	Data Declarations for CMPP3 and CMPP4
(8)	694	VAX\$MOVP - Move Packed
(9)	808	Routine to Strip Leading Zeros from Decimal String
(10)	947	DECIMAL_ROPRAND
(11)	980	DECIMAL_ACCVIO - Reflect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

```
0000 1 .TITLE VAX$DECIMAL - VAX-11 Packed Decimal Instruction Emulator
0000 2 .IDENT /V04-000/
0000 3
0000 4
0000 5 *****
0000 6
0000 7 *
0000 8 * COPYRIGHT (c) 1978, 1980, 1982, 1984 BY
0000 9 * DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.
0000 10 * ALL RIGHTS RESERVED.
0000 11 *
0000 12 * THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED
0000 13 * ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE AND WITH THE
0000 14 * INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR ANY OTHER
0000 15 * COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY
0000 16 * OTHER PERSON. NO TITLE TO AND OWNERSHIP OF THE SOFTWARE IS HEREBY
0000 17 * TRANSFERRED.
0000 18 *
0000 19 * THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT NOTICE
0000 20 * AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT
0000 21 * CORPORATION.
0000 22 *
0000 23 * DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF ITS
0000 24 * SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.
0000 25 *
0000 26 *****
0000 27
0000 28
0000 29 ++
0000 30 Facility:
0000 31
0000 32 VAX-11 Instruction Emulator
0000 33
0000 34 Abstract:
0000 35
0000 36 The routines in this module emulate the VAX-11 packed decimal
0000 37 instructions. These procedures can be a part of an emulator
0000 38 package or can be called directly after the input parameters
0000 39 have been loaded into the architectural registers.
0000 40
0000 41 The input parameters to these routines are the registers that
0000 42 contain the intermediate instruction state.
0000 43
0000 44 Environment:
0000 45
0000 46 These routines run at any access mode, at any IPL, and are AST
0000 47 reentrant.
0000 48
0000 49 Author:
0000 50
0000 51 Lawrence J. Kenah
0000 52
0000 53 Creation Date
0000 54
0000 55 24 September 1982
0000 56
0000 57 Modified by:
```

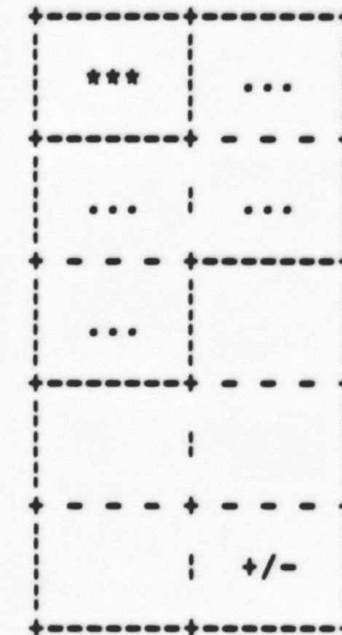

0000	58	:	
0000	59	:	
0000	60	:	V01-007 LJK0043 Lawrence J. Kenah 26-Jul-1984
0000	61	:	Change STRIP ZEROS routines so that they are more forgiving
0000	62	:	when confronted with poorly formed packed decimal strings.
0000	63	:	Specifically, do not allow string lengths smaller than zero.
0000	64	:	
0000	65	:	V01-006 LJK0038 Lawrence J. Kenah 19-Jul-1984
0000	66	:	Insure that initial setting of C-bit is preserved when
0000	67	:	MOVP is restarted after an access violation.
0000	68	:	
0000	69	:	V01-005 LJK0024 Lawrence J. Kenah 20-Feb-1984
0000	70	:	Add code to handle access violations.
0000	71	:	
0000	72	:	V01-004 LJK0013 Lawrence J. Kenah 17-Nov-1983
0000	73	:	Move CVTPL to a separate module.
0000	74	:	
0000	75	:	V01-003 LJK0008 Lawrence J. Kenah 18-Oct-1983
0000	76	:	Move decimal arithmetic and numeric string routines to
0000	77	:	separate modules.
0000	78	:	
0000	79	:	V01-002 LJK0006 Lawrence J. Kenah 14-Oct-1983
0000	80	:	Fix code that handles arithmetic traps. Add reserved operand
0000	81	:	processing. Add PROBEs and other code to handle access
0000	82	:	violations.
0000	83	:	
0000	84	:	V01-001 Original Lawrence J. Kenah 24-Sep-82
0000	85	:	--

0000 87 .SUBTITLE Miscellaneous Notes
0000 88
0000 89 :+
0000 90 : There are several techniques that are used throughout the routines in this
0000 91 : module that are worth a comment somewhere. Rather than duplicate near
0000 92 : identical commentary in several places, we will describe these general
0000 93 : techniques in a single place.
0000 94 :
0000 95 : 1. The VAX-11 architecture specifies that several kinds of input produce
0000 96 : UNPREDICTABLE results. They are:
0000 97 :
0000 98 : o Illegal decimal digit in packed decimal string
0000 99 :
0000 100 : o Illegal sign specifier (other than 10 through 15) in low nibble of
0000 101 : highest addressed byte of packed decimal string
0000 102 :
0000 103 : o Packed decimal string with even number of digits that contains
0000 104 : other than a zero in the high nibble of the lowest addressed byte
0000 105 :
0000 106 : These routines take full advantage of the meaning of UNPREDICTABLE.
0000 107 : In general, the code assumes that all input is correct. The operation
0000 108 : of the code for illegal input is not even consistent but is simply
0000 109 : whatever happens to be convenient in a particular place.
0000 110 :
0000 111 : 2. All of these routines accumulate information about condition codes at
0000 112 : several key places in a routine. This information is kept in a
0000 113 : register (usually R11) that is used to set the final condition codes
0000 114 : in the PSW. In order to allow the register to obtain its correct
0000 115 : contents when the routine exits (without further affecting the
0000 116 : condition codes), the condition codes are set from the register
0000 117 : (BISPSW reg) and the register is then restored with a POPR
0000 118 : instruction, which does not affect condition codes.
0000 119 :
0000 120 : 3. There are several instances in these routines where it is necessary to
0000 121 : determine the difference in length between an input and an output
0000 122 : string and perform special processing on the excess digits. When the
0000 123 : longer string is a packed decimal string (it does not matter if the
0000 124 : packed decimal string is an input string or an output string), it is
0000 125 : sometimes useful to convert the difference in digits to a byte count.
0000 126 :
0000 127 : There are four different cases that exist. We will divide these cases
0000 128 : into two sets of two cases, depending on whether the shorter length is
0000 129 : even or odd.
0000 130 :
0000 131 : In the pictures that appear below, a blank box indicates a digit in
0000 132 : the shorter string. A string of three dots in a box indicates a digit
0000 133 : in the longer string. A string of three stars indicates an unused
0000 134 : digit in a decimal string. The box that contains +/- obviously
0000 135 : indicates the sign nibble in a packed decimal string.
0000 136 :
0000 137 : (cont.)

0000 139 :
0000 140 :
0000 141 :
0000 142 :
0000 143 :
0000 144 :
0000 145 :
0000 146 :
0000 147 :
0000 148 :
0000 149 :
0000 150 :
0000 151 :
0000 152 :
0000 153 :
0000 154 :
0000 155 :
0000 156 :
0000 157 :
0000 158 :
0000 159 :
0000 160 :
0000 161 :
0000 162 :
0000 163 :
0000 164 :
0000 165 :
0000 166 :
0000 167 :
0000 168 :
0000 169 :
0000 170 :
0000 171 :
0000 172 :
0000 173 :
0000 174 :
0000 175 :
0000 176 :
0000 177 :
0000 178 :
0000 179 :
0000 180 :
0000 181 :
0000 182 :
0000 183 :
0000 184 :
0000 185 :
0000 186 :
0000 187 :
0000 188 :
0000 189 :
0000 190 :
0000 191 :
0000 192 :
0000 193 : (cont.)

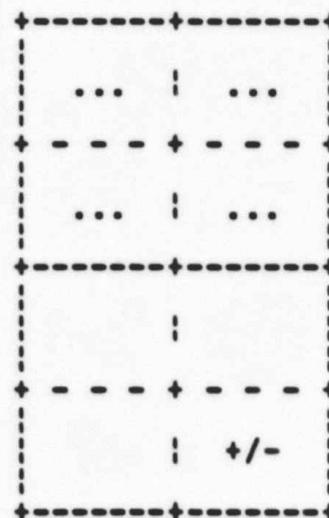


A Longer string odd
Difference odd

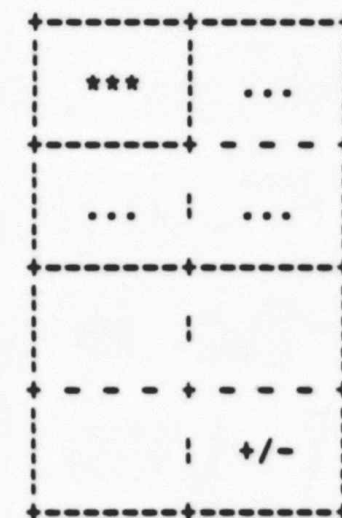


B Longer string even
Difference even

CASE 1 Shorter string has even number of digits



A Longer string odd
Difference even



B Longer string even
Difference odd

CASE 2 Shorter string has odd number of digits

0000 195 :
0000 196 :
0000 197 :
0000 198 :
0000 199 :
0000 200 :
0000 201 :
0000 202 :
0000 203 :
0000 204 :
0000 205 :
0000 206 :
0000 207 :
0000 208 :
0000 209 :
0000 210 :
0000 211 :
0000 212 :
0000 213 :
0000 214 :
0000 215 :
0000 216 :
0000 217 :
0000 218 :
0000 219 :
0000 220 :
0000 221 :
0000 222 :
0000 223 :
0000 224 :
0000 225 :
0000 226 :
0000 227 :
0000 228 :
0000 229 :
0000 230 :
0000 231 :
0000 232 :
0000 233 :
0000 234 : 4.
0000 235 :
0000 236 :
0000 237 :
0000 238 :
0000 239 :
0000 240 :-

In general, the code must calculate the number of bytes that contain the excess digits. Most of the time, the interesting number includes complete excess bytes. The excess digit in the high nibble of the highest addressed byte (both parts of Case 1) is ignored.

In three out of four cases, the difference (called R5 from this point on) can be simply divided by two to obtain a byte count. In one case (Case 2 B), this is not correct. (For example, $3/2 = 1$ and we want to get a result of 2.) Note, however, that in both parts of Case 2, we can add 1 to R5 before we divide by two. In Case 2 B, this causes the result to be increased by 1, which is what we want. In Case 2 A, because the original difference is even, an increment of one before we divide by two has no effect on the final result.

The correct code sequence to distinguish case 2 B from the other three cases involves two BLBx instructions. A simpler sequence that accomplishes correct results in all four cases when converting a digit count to a byte count is something like

```
BLBC    length-of-shorter,10$  
INCL    R5  
10$:    ASHL    #-1,R5,R5
```

where the length of the shorter string will typically be contained in either R0 or R2.

Note that we could also look at both B parts, performing the extra INCL instruction when the longer string is even. In case 1 B, this increment transforms an even difference to an odd number but does not affect the division by two. In case 2 B, the extra increment produces the correct result. This option is not used in these routines.

The two routines for CVTSP and CVTTP need a slightly different number. They want the number of bytes including the byte containing the excess high nibble. For Case 2, the above calculation is still valid. For Case 1, it is necessary to add one to R5 after the R5 is divided by two to obtain the correct byte count.

There is a routine called STRIP_ZEROS that removes high order zeros from decimal strings. This routine is not used by all of the routines in this module but only by those routines that perform complicated calculations on each byte of the input string. For these routines, the overhead of testing for and discarding leading zeros is less than the more costly per byte overhead of these routines.


```
0000 242      .SUBTITLE      Declarations
0000 243
0000 244 ; Include files:
0000 245
0000 246      .NOCROSS
0000 247      .ENABLE      SUPPRESSION      ; No cross reference for these
0000 248                                         ; No symbol table entries either
0000 249      CMPP3_DEF      ; Bit fields in CMPP3 registers
0000 250      CMPP4_DEF      ; Bit fields in CMPP4 registers
0000 251      MOVP_DEF      ; Bit fields in MOVP registers
0000 252
0000 253      PACK_DEF      ; Stack usage by exception handler
0000 254      STACK_DEF      ; Stack usage of original exception
0000 255
0000 256      $PSLDEF      ; Define bit fields in PSL
0000 257
0000 258      .DISABLE      SUPPRESSION      ; Turn on symbol table again
0000 259      .CROSS      ; Cross reference is OK now
0000 260
0000 261 ; External declarations
0000 262
0000 263      .DISABLE      GLOBAL
0000 264
0000 265      .EXTERNAL -
0000 266      VAX$REFLECT_FAULT,-
0000 267      VAX$ROPRAND
0000 268
0000 269 ; PSECT Declarations:
0000 270
0000 271      .DEFAULT      DISPLACEMENT , WORD
0000 272
00000000 273      .PSECT _VAX$CODE PIC, USR, CON, REL, LCL, SHR, EXE, RD, NOWRT, LONG
0000 274
0000 275      BEGIN_MARK_POINT
```



```
0000 277 .SUBTITLE Conversion Tables
0000 278
0000 279 :+
0000 280 : The following tables are designed to perform fast conversions between
0000 281 : numbers in the range 0 to 99 and their decimal equivalents. The tables
0000 282 : are used by placing the input parameter into a register and then using
0000 283 : the contents of that register as an index into the table.
0000 284 :-
0000 285
0000 286 :+
0000 287 : Decimal Digits to Binary Number
0000 288 :
0000 289 : The following table is used to convert a packed decimal byte to its binary
0000 290 : equivalent.
0000 291 :
0000 292 : Packed decimal numbers that contain illegal digits in the low nibble
0000 293 : convert as if the low nibble contained a zero. That is, the binary number
0000 294 : will be a multiple of ten. This is done so that this table can be used to
0000 295 : convert the least significant (highest addressed) byte of a decimal string
0000 296 : without first masking off the sign "digit".
0000 297 :
0000 298 : Illegal digits in the high nibble produce UNPREDICTABLE results because the
0000 299 : table does not contain entries to handle these illegal constructs.
0000 300 :-
0000 301
0000 302 : Binary Equivalent
0000 303 : -----
0000 304
0000 305 DECIMAL$PACKED_TO_BINARY_TABLE::
0000 306
0000 307 .BYTE 00 , 01 , 02 , 03 , 04 , - ; Index ^X00
000A 308 05 , 06 , 07 , 08 , 09 , - ; to ^X09
000A 309
000A 310 .BYTE 00 , 00 , 00 , 00 , 00 , 00 ; Illegal decimal digits
0010 311
0010 312 .BYTE 10 , 11 , 12 , 13 , 14 , - ; Index ^X10
001A 313 15 , 16 , 17 , 18 , 19 , - ; to ^X19
001A 314
001A 315 .BYTE 10 , 10 , 10 , 10 , 10 , 10 ; Illegal decimal digits
0020 316
0020 317 .BYTE 20 , 21 , 22 , 23 , 24 , - ; Index ^X20
002A 318 25 , 26 , 27 , 28 , 29 , - ; to ^X29
002A 319
002A 320 .BYTE 20 , 20 , 20 , 20 , 20 , 20 ; Illegal decimal digits
0030 321
0030 322 .BYTE 30 , 31 , 32 , 33 , 34 , - ; Index ^X30
003A 323 35 , 36 , 37 , 38 , 39 , - ; to ^X39
003A 324
003A 325 .BYTE 30 , 30 , 30 , 30 , 30 , 30 ; Illegal decimal digits
0040 326
0040 327 .BYTE 40 , 41 , 42 , 43 , 44 , - ; Index ^X40
004A 328 45 , 46 , 47 , 48 , 49 , - ; to ^X49
004A 329
004A 330 .BYTE 40 , 40 , 40 , 40 , 40 , 40 ; Illegal decimal digits
0050 331
0050 332 .BYTE 50 , 51 , 52 , 53 , 54 , - ; Index ^X50
005A 333 55 , 56 , 57 , 58 , 59 , - ; to ^X59
```


32 32 32 32 32 32	005A 334		
	005A 335	.BYTE	50 , 50 , 50 , 50 , 50 , 50 ; Illegal decimal digits
45 44 43 42 41 40 3F 3E 3D 3C	0060 336		
	0060 337	.BYTE	60 , 61 , 62 , 63 , 64 , - ; Index ^X60
	006A 338		65 , 66 , 67 , 68 , 69 , - ; to ^X69
	006A 339		
3C 3C 3C 3C 3C 3C	006A 340	.BYTE	60 , 60 , 60 , 60 , 60 , 60 ; Illegal decimal digits
	0070 341		
4F 4E 4D 4C 4B 4A 49 48 47 46	0070 342	.BYTE	70 , 71 , 72 , 73 , 74 , - ; Index ^X70
	007A 343		75 , 76 , 77 , 78 , 79 , - ; to ^X79
	007A 344		
46 46 46 46 46 46	007A 345	.BYTE	70 , 70 , 70 , 70 , 70 , 70 ; Illegal decimal digits
	0080 346		
59 58 57 56 55 54 53 52 51 50	0080 347	.BYTE	80 , 81 , 82 , 83 , 84 , - ; Index ^X80
	008A 348		85 , 86 , 87 , 88 , 89 , - ; to ^X89
	008A 349		
50 50 50 50 50 50	008A 350	.BYTE	80 , 80 , 80 , 80 , 80 , 80 ; Illegal decimal digits
	0090 351		
63 62 61 60 5F 5E 5D 5C 5B 5A	0090 352	.BYTE	90 , 91 , 92 , 93 , 94 , - ; Index ^X90
	009A 353		95 , 96 , 97 , 98 , 99 , - ; to ^X99
	009A 354		
5A 5A 5A 5A 5A 5A	009A 355	.BYTE	90 , 90 , 90 , 90 , 90 , 90 ; Illegal decimal digits

Binary Number to Decimal Equivalent

The following table is used to do a fast conversion from a binary number stored in a byte to its decimal representation. The table structure assumes that the number lies in the range 0 to 99. Numbers that lie outside this range produce UNPREDICTABLE results.

Decimal Equivalents

Binary

	DECIMAL\$BINARY_TO_PACKED_TABLE::		
09 08 07 06 05 04 03 02 01 00	00A0 360	.BYTE	^X00 , ^X01 , ^X02 , ^X03 , ^X04 , - ; 0 through 9
	00AA 361		^X05 , ^X06 , ^X07 , ^X08 , ^X09 , - ;
19 18 17 16 15 14 13 12 11 10	00AA 362	.BYTE	^X10 , ^X11 , ^X12 , ^X13 , ^X14 , - ; 10 through 19
	00B4 363		^X15 , ^X16 , ^X17 , ^X18 , ^X19 , - ;
29 28 27 26 25 24 23 22 21 20	00B4 364	.BYTE	^X20 , ^X21 , ^X22 , ^X23 , ^X24 , - ; 20 through 29
	00BE 365		^X25 , ^X26 , ^X27 , ^X28 , ^X29 , - ;
39 38 37 36 35 34 33 32 31 30	00BE 366	.BYTE	^X30 , ^X31 , ^X32 , ^X33 , ^X34 , - ; 30 through 39
	00C8 367		^X35 , ^X36 , ^X37 , ^X38 , ^X39 , - ;
49 48 47 46 45 44 43 42 41 40	00C8 368	.BYTE	^X40 , ^X41 , ^X42 , ^X43 , ^X44 , - ; 40 through 49
	00D2 369		^X45 , ^X46 , ^X47 , ^X48 , ^X49 , - ;
59 58 57 56 55 54 53 52 51 50	00D2 370	.BYTE	^X50 , ^X51 , ^X52 , ^X53 , ^X54 , - ; 50 through 59
	00DC 371		^X55 , ^X56 , ^X57 , ^X58 , ^X59 , - ;
69 68 67 66 65 64 63 62 61 60	00DC 372	.BYTE	^X60 , ^X61 , ^X62 , ^X63 , ^X64 , - ; 60 through 69
	00E6 373		^X65 , ^X66 , ^X67 , ^X68 , ^X69 , - ;

79 78 77 76 75 74 73 72 71 70	00E6 391 00E6 392 00F0 393 00F0 394	.BYTE ^X70 ; ^X71 ; ^X72 ; ^X73 ; ^X74 , - ; 70 through 79 ^X75 ; ^X76 ; ^X77 ; ^X78 ; ^X79 ;
89 88 87 86 85 84 83 82 81 80	00F0 395 00FA 396 00FA 397	.BYTE ^X80 ; ^X81 ; ^X82 ; ^X83 ; ^X84 , - ; 80 through 89 ^X85 ; ^X86 ; ^X87 ; ^X88 ; ^X89 ;
99 98 97 96 95 94 93 92 91 90	00FA 398 0104 399 0104 400	.BYTE ^X90 ; ^X91 ; ^X92 ; ^X93 ; ^X94 , - ; 90 through 99 ^X95 ; ^X96 ; ^X97 ; ^X98 ; ^X99 ;


```
0104 402 .SUBTITLE VAX$CMPPx - Compare Packed
0104 403 :+
0104 404 : Functional Description:
0104 405 :
0104 406 : In 3 operand format, the source 1 string specified by the length and
0104 407 : source 1 address operands is compared to the source 2 string specified
0104 408 : by the length and source 2 address operands. The only action is to
0104 409 : affect the condition codes.
0104 410 :
0104 411 : In 4 operand format, the source 1 string specified by the source 1
0104 412 : length and source 1 address operands is compared to the source 2 string
0104 413 : specified by the source 2 length and source 2 address operands. The
0104 414 : only action is to affect the condition codes.
0104 415 :
0104 416 : Input Parameters:
0104 417 :
0104 418 : Entry at VAX$CMPP3
0104 419 :
0104 420 : R0 - len.rw Length of either decimal string
0104 421 : R1 - src1addr.ab Address of first packed decimal string
0104 422 : R3 - src2addr.ab Address of second packed decimal string
0104 423 :
0104 424 : Entry at VAX$CMPP4
0104 425 :
0104 426 : R0 - src1len.rw Length of first packed decimal string
0104 427 : R1 - src1addr.ab Address of first packed decimal string
0104 428 : R2 - src2len.rw Length of second packed decimal string
0104 429 : R3 - src2addr.ab Address of second packed decimal string
0104 430 :
0104 431 : Output Parameters:
0104 432 :
0104 433 : R0 = 0
0104 434 : R1 = Address of the byte containing the most significant digit of
0104 435 : the first source string
0104 436 : R2 = 0
0104 437 : R3 = Address of the byte containing the most significant digit of
0104 438 : the second source string
0104 439 :
0104 440 : Condition Codes:
0104 441 :
0104 442 : N <- first source string LSS second source string
0104 443 : Z <- first source string EQL second source string
0104 444 : V <- 0
0104 445 : C <- 0
0104 446 :
0104 447 : Register Usage:
0104 448 :
0104 449 : This routine uses R0 through R5. The condition codes are recorded
0104 450 : in R2 as the routine executes.
0104 451 :
0104 452 : Algorithm:
0104 453 :
0104 454 : TBS
0104 455 : -
0104 456 :
0104 457 .SUBTITLE Data Declarations for CMPP3 and CMPP4
0104 458
```



```
0104 459 :+
0104 460 : Define some bit fields that allow recording the presence of minus signs
0104 461 : in either or both of the source strings.
0104 462 :-
0104 463
0104 464 $DEFINI CMPPx_FLAGS
0000 465
0000 466 _VIELD CMPPx,0,<-
0000 467 <SRC1_MINUS,,M>,-
0000 468 <SRC2_MINUS,,M>,-
0000 469 >
0000 470
0000 471 $DEFEND CMPPx_FLAGS
0104 472
0104 473 .ENABLE LOCAL_BLOCK
0104 474
0104 475 VAX$CMPP3::
52 50 3C 0104 476 MOVZWL R0,R2 ; Make two source lengths equal
0B 11 0107 477 BRB 10$ ; Only make one length check
0109 478
0109 479 VAX$CMPP4::
0109 480 ROPRAND_CHECK R2 ; Insure that R2 LEQU 31
0114 481 10$: ROPRAND_CHECK R0 ; Insure that R0 LEQU 31
043F 8F BB 011C 482 PUSHF #M<R0,R1,R2,R3,R4,R5,R10> ; Save some registers
0120 483 ESTABLISH_HANDLER - ; Store address of access
0120 484 DECIMAL_ACCVIO ; violation handler
0125 485
0125 486 : Get sign of first input string
0125 487
55 50 04 54 D4 0125 488 CLRL R4 ; Assume both strings contain '+'
EF 0127 489 EXTZV #1,#4,R0,R5 ; Convert digit count to byte count
012C 490 MARK_POINT CMPPx_ACCVIO
55 6145 F0 8F 8B 012C 491 BICB3 #B11110000,(R1)[R5],R5 ; R5 contains "sign" digit
0132 492 CASE R5,TYPE=B,LIMIT=#10,<- ; Dispatch on sign digit
0132 493 30$,- ; 10 => sign is '+'
0132 494 20$,- ; 11 => sign is '-'
0132 495 30$,- ; 12 => sign is '+'
0132 496 20$,- ; 13 => sign is '-'
0132 497 30$,- ; 14 => sign is '+'
0132 498 30$,- ; 15 => sign is '+'
0132 499 >
54 01 C8 0142 501 20$: BISL2 #CMPPx_M_SRC1_MINUS,R4 ; Remember that src1 contains '-'
0145 502
0145 503 : Now get sign of second input string
0145 504
55 52 04 01 EF 0145 505 30$: EXTZV #1,#4,R2,R5 ; Convert digit count to byte count
014A 506 MARK_POINT CMPPx_ACCVIO
55 6345 F0 8F 8B 014A 507 BICB3 #B11110000,(R3)[R5],R5 ; R5 contains "sign" digit
0150 508 CASE R5,TYPE=B,LIMIT=#10,<- ; Dispatch on sign digit
0150 509 50$,- ; 10 => sign is '+'
0150 510 40$,- ; 11 => sign is '-'
0150 511 50$,- ; 12 => sign is '+'
0150 512 40$,- ; 13 => sign is '-'
0150 513 50$,- ; 14 => sign is '+'
0150 514 50$,- ; 15 => sign is '+'
0150 515 >
```



```
54 02 C8 0160 516 40$: BISL2 #CMPPx_M_SRC2_MINUS,R4 ; Remember that src2 contains '-'
0160 517
0163 518
0163 519 ; At this point, we have determined the signs of both input strings. If the
0163 520 ; strings have different signs, then the comparison is done except for the
0163 521 ; extraordinary case of comparing a minus zero to a plus zero. If both signs
0163 522 ; are the same, then a digit-by-digit comparison is required.
0163 523
0163 524 50$: CASE R4,LIMIT=#0,TYPE=B,<- ; Dispatch on combination of signs
0163 525 60$,- ; Both signs are '+'
0163 526 MINUS_ZERO_CHECK,- ; Signs are different
0163 527 MINUS_ZERO_CHECK,- ; Signs are different
0163 528 60$,- ; Both signs are '-'
0163 529 >
016F 530
016F 531 ; Both strings have the same sign. If the strings have different lengths, then
016F 532 ; the excess digits in the longer string are checked for nonzero because that
016F 533 ; eliminates the need for further comparison.
016F 534
55 52 50 C3 016F 535 60$: SUBL3 R0,R2,R5 ; Get difference in lengths
30 13 0173 536 BEQL EQUAL_LENGTH ; Strings have the same size
15 19 0175 537 BLSS SRC2_SHORTER ; src2 is shorter than src1
0177 538
0177 539 ; This code executes when src1 is shorter than src2. That is, R0 LSSU R2.
0177 540 ; The large comment at the beginning of this module explains the need for the
0177 541 ; INCL R5 instruction when R0, the length of the shorter string, is odd.
0177 542
0177 543 SRC1_SHORTER:
0177 544 BLBC R0,70$ ; Skip adjustment if R0 is even
55 55 04 01 EF 017A 545 INCL R5 ; Adjust digit difference if R0 is odd
22 13 017C 546 70$: EXTZV #1,#4,R5,R5 ; Convert digit count to byte count
0181 547 BEQL EQUAL_LENGTH ; Skip loop if no entire bytes in excess
0183 548
0183 549 MARK_POINT CMPPx_ACCVIO
0183 550 80$: TSTB (R3)+ ; Test excess src2 digits for nonzero
55 12 0185 551 BNEQ SRC1_SMALLER ; All done if nonzero. src1 LSS src2
F9 55 F5 0187 552 SOBGTR R5,80$ ; Test for end of loop
018A 553
19 11 018A 554 BRB EQUAL_LENGTH ; Enter loop that performs comparison
018C 555
018C 556 ; This code executes when src2 is shorter than src1. That is, R2 LSSU R0.
018C 557 ; The large comment at the beginning of this module explains the need for the
018C 558 ; INCL R5 instruction when R2, the length of the shorter string, is odd.
018C 559
018C 560 SRC2_SHORTER:
50 52 D0 018C 561 MOVL R2,R0 ; R0 contains number of remaining digits
55 55 CE 018F 562 MNEGL R5,R5 ; Make difference positive
02 52 E9 0192 563 BLBC R2,90$ ; Skip adjustment if R2 is even
55 55 D6 0195 564 INCL R5 ; Adjust digit difference if R2 is odd
55 55 04 01 EF 0197 565 90$: EXTZV #1,#4,R5,R5 ; Convert digit count to byte count
07 13 019C 566 BEQL EQUAL_LENGTH ; Skip loop if no entire bytes in excess
019E 567
019E 568 MARK_POINT CMPPx_ACCVIO
019E 569 100$: TSTB (R1)+ ; Test excess src1 digits for nonzero
32 12 01A0 570 BNEQ SRC2_SMALLER ; All done if nonzero. src2 LSS src1
F9 55 F5 01A2 571 SOBGTR R5,100$ ; Test for end of loop
01A5 572
```



```

50 50 04 01 EF 01A5 573 ; All excess digits are zero. We must now perform a digit-by-digit comparison
      08 13 01A5 574 ; of the remaining digits in the two strings. R0 contains the remaining number
      01AA 575 ; of digits in either string.
      01AC 576
      01AC 577 EQUAL_LENGTH:
      83 81 91 01A5 578 EXTZV #1,#4,R0,R0 ; Convert digit count to byte count
      21 12 01AA 579 BEQL 120$ ; All done if no digits remain
      F8 50 F5 01AC 580
      01AC 581 MARK_POINT CMPPx_ACCVIO
      01AC 582 110$: CMPB (R1)+,(R3)+ ; Compare next two digits
      01AF 583 BNEQ NOT_EQUAL ; Comparison complete if not equal
      01B1 584 SOBGTR R0,T10$ ; Test for end of loop
      01B4 585
      01B4 586 ; Compare least significant digit in source and destination strings
      01B4 587
      51 61 0F 8B 01B4 588 MARK_POINT CMPPx_ACCVIO
      01B8 589 120$: BICB3 #^B00001111,(R1),R1 ; Strip sign from last src1 digit
      53 63 0F 8B 01B8 590 MARK_POINT CMPPx_ACCVIO
      53 51 91 01BC 591 BICB3 #^B00001111,(R3),R3 ; Strip sign from last src2 digit
      11 12 01BF 592 CMPB R1,R3 ; Compare least significant digits
      01C1 593 BNEQ NOT_EQUAL
      01C1 594
      01C1 595 ; At this point, all tests have been exhausted and the two strings have
      01C1 596 ; been shown to be equal. Set the Z-bit, clear the remaining condition
      01C1 597 ; codes, and restore saved registers.
      01C1 598
      52 04 9A 01C1 599 SRC1_EQL_SRC2:
      01C4 600 MOVZBL #PSLSM_Z,R2 ; Set condition codes for src1 EQL src2
      01C4 601
      01C4 602 ; This is the common exit path. R2 contains the appropriate settings for the
      01C4 603 ; N- and Z-bits. There is no other expected input at this point.
      01C4 604
      01C4 605 CMPPx_EXIT:
      08 6E D4 01C4 606 CLRL (SP) ; Set saved R0 to 0
      08 AE D4 01C6 607 CLRL 8(SP) ; Set saved R2 to 0
      0F B9 01C9 608 BICPSW #<PSLSM_N!PSLSM_Z!PSLSM_V!PSLSM_C> ; Start with clean slate
      52 B8 01CB 609 BISPSW R2 ; Set N- and Z-bits as appropriate
      043F 8F BA 01CD 610 POPR #^M<R0,R1,R2,R3,R4,R5,R10> ; Restore saved registers
      05 01D1 611 RSB ; Return
      01D2 612
      01D2 613 ; The following code executes if specific digits in the two strings have
      01D2 614 ; tested not equal. Separate pieces of code are selected for the two
      01D2 615 ; different cases of not equal. Note that unsigned comparisons are required
      01D2 616 ; here because the decimal digits '8' and '9', when appearing in the high
      01D2 617 ; nibble, can cause the sign bit to be set.
      01D2 618
      08 01D2 619 NOT_EQUAL:
      1F 01D2 620 BLSSU SRC1_SMALLER ; Branch if src1 is smaller than src2
      01D4 621
      01D4 622 ; The src2 string has a smaller magnitude than the src1 string. The setting
      01D4 623 ; of the signs determines how this transforms to a signed comparison. That is,
      01D4 624 ; if both input signs are minus, then reverse the sense of the comparison.
      01D4 625
      08 54 01 E0 01D4 626 SRC2_SMALLER:
      01D8 627 BBS #CMPPx_V_SRC2_MINUS,R4,SRC1_SMALLER_REALLY
      01D8 628
      01D8 629 ; The SRC2 string has been determined to be smaller than the SRC1 string
```



```

01D8 630 SRC2_SMALLER REALLY:
01D8 631
52 D4 01D8 632 CLRL R2 ; Clear both N- and Z-bits
E8 11 01DA 633 BRB CMPPx_EXIT ; Join common exit code
01DC 634
01DC 635 ; The src1 string has a smaller magnitude than the src2 string. The setting
01DC 636 ; of the signs determines how this transforms to a signed comparison. That is,
01DC 637 ; if both input signs are minus, then reverse the sense of the comparison.
01DC 638
01DC 639 SRC1_SMALLER:
F8 54 00 E0 01DC 640 BBS #CMPPx_V_SRC1_MINUS,R4,SRC2_SMALLER REALLY
01E0 641
01E0 642 ; The src1 string has been determined to be smaller than the src2 string
01E0 643
01E0 644 SRC1_SMALLER REALLY:
52 08 90 01E0 645 MOVB #PSLSM_N,R2 ; Clear both N- and Z-bits
DF 11 01E3 646 BRB CMPPx_EXIT ; Join common exit code
01E5 647
01E5 648 ; The following code executes if the two input strings have different
01E5 649 ; signs. We need to determine if a comparison between plus zero and minus
01E5 650 ; zero is being made, because such a comparison should test equal. We scan
01E5 651 ; first one string and then the other. If we find a nonzero digit anywhere
01E5 652 ; along the way, we immediately exit this test and set the final condition
01E5 653 ; codes such that the "-" string is smaller than the "+" string. If we
01E5 654 ; exhaust both strings without finding a nonzero digit, then we report
01E5 655 ; that the two strings are equal.
01E5 656
01E5 657 MINUS_ZERO CHECK:
50 50 04 01 EF 01E5 658 EXTZV #1,#4,R0,R0 ; Convert src1 digit count to byte count
07 13 01EA 659 BEQL 170$ ; Skip loop if only single digit
01EC 660
01EC 661 MARK_POINT CMPPx_ACCVIO
81 95 01EC 662 160$: TSTB (R1)+ ; Test next byte for nonzero
1D 12 01EE 663 BNEQ CMPPx_NOT_ZERO ; Exit loop if nonzero
F9 50 F5 01F0 664 SOBGTR R0,160$ ; Test for end of loop
01F3 665
01F3 666 MARK_POINT CMPPx_ACCVIO
61 F0 8F 93 01F3 667 170$: BITB #^B11110000,(R1) ; Test least significant digit
14 12 01F7 668 BNEQ CMPPx_NOT_ZERO ; Exit if this digit is not zero
01F9 669
01F9 670 ; All digits in src1 are zero. Now we must look for nonzero digits in src2.
01F9 671
52 52 04 01 EF 01F9 672 EXTZV #1,#4,R2,R2 ; Convert src2 digit count to byte count
07 13 01FE 673 BEQL 190$ ; Skip loop if only single digit
0200 674
0200 675 MARK_POINT CMPPx_ACCVIO
83 95 0200 676 180$: TSTB (R3)+ ; Test next byte for nonzero
09 12 0202 677 BNEQ CMPPx_NOT_ZERO ; Exit loop if nonzero
F9 52 F5 0204 678 SOBGTR R2,180$ ; Test for end of loop
0207 679
0207 680 MARK_POINT CMPPx_ACCVIO
63 F0 8F 93 0207 681 190$: BITB #^B11110000,(R3) ; Test least significant digit
B4 13 020B 682 BEQL SRC1_EQL_SRC2 ; Branch if two strings are equal
020D 683
020D 684 ; At least one of the two input strings contains at least one nonzero digit.
020D 685 ; That knowledge is sufficient to determine the result of the comparison
020D 686 ; based simply on the two (necessarily different) signs of the input strings.

```

. . .
 CMP
 CMP
 CMP
 CMP
 CMP
 CMP
 CMP
 DEC
 DEC
 DEC
 DEC
 DEC
 DEC
 EQU
 HAN
 MIN
 MOD
 MOV
 MOV
 MOV
 MOV
 MOV
 MOV
 NOT
 PAC
 PAC
 PAC
 PAC
 PC
 PSL
 PSL
 PSL
 PSL
 PSL
 SIZ
 SRC
 SRC
 SRC
 SRC
 SRC
 SRC
 SRC
 TAB
 VAX
 VAX
 VAX
 VAX
 VAX

			020D	687	
			020D	688	CMPPx_NOT ZERO:
C7 54	01	E0	020D	689	BBS #CMPPx V SRC2 MINUS,R4,SRC2_SMALLER_REALLY
	CD	11	0211	690	BRB SRC1_SMALLER_REALLY
			0213	691	
			0213	692	.DISABLE LOCAL_BLOCK


```
0213 694 .SUBTITLE VAX$MOVP - Move Packed
0213 695 :+
0213 696 : Functional Description:
0213 697 :
0213 698 : The destination string specified by the length and destination address
0213 699 : operands is replaced by the source string specified by the length and
0213 700 : source address operands.
0213 701 :
0213 702 : Input Parameters:
0213 703 :
0213 704 : R0 - len.rw Length of input and output decimal strings
0213 705 : R1 - srcaddr.ab Address of input packed decimal string
0213 706 : R3 - dstaddr.ab Address of output packed decimal string
0213 707 :
0213 708 : PSL<C> Contains setting of C-bit when MOVP executed
0213 709 :
0213 710 : Output Parameters:
0213 711 :
0213 712 : R0 = 0
0213 713 : R1 = Address of byte containing most significant digit of
0213 714 : the source string
0213 715 : R2 = 0
0213 716 : R3 = Address of byte containing most significant digit of
0213 717 : the destination string
0213 718 :
0213 719 : Condition Codes:
0213 720 :
0213 721 : N <- destination string LSS 0
0213 722 : Z <- destination string EQL 0
0213 723 : V <- 0
0213 724 : C <- C ; Note that C-bit is preserved!
0213 725 :
0213 726 : Register Usage:
0213 727 :
0213 728 : This routine uses R0 through R3. The condition codes are recorded
0213 729 : in R2 as the routine executes.
0213 730 :
0213 731 : Notes:
0213 732 :
0213 733 : The initial value of the C-bit must be captured (saved in R2) before
0213 734 : any instructions execute that alter the C-bit.
0213 735 :-
0213 736 :
0213 737 VAX$MOVP::
0213 738 MOVPSL R2 ; Save initial PSL (to preserve C-bit)
0215 739
0215 740 ASSUME MOVP_B_STATE EQ 2 ; Make sure that FPD bit is in R0<23:16>
0215 741
0215 742 BBC #<MOVP_V_FPD + 16>,R0,5$ ; Branch if first time
0219 743 EXTZV #<MOVP_V_SAVED_PSW + 16>,- ; Otherwise, replace condition
021B 744 #MOVP_S_SAVED_PSW,R0,R2 ; codes with previous settings
021E 745
021E 746 5$: ROPRAND_CHECK R0 ; Insure that R0 LEQU 31
0229 747
0229 748 ; Save the starting addresses of the input and output strings in addition to
0229 749 ; the digit count operand (initial R0 contents.) Store a place holder for
0229 750 ; saved R2.
```

52 DC 0213 738

05 50 14 E1 0215 742

52 50 04 EF 0219 743

```
040F 8F BB 0229 751
0229 752 PUSHR #^M<R0,R1,R2,R3,R10> ; Save initial register contents
022D 753 ESTABLISH_HANDLER - ; Store address of access
022D 754 DECIMAL_ACCVIO ; violation handler
0232 755
52 03 01 02 F0 0232 756 INSV #<PSLSM_Za-1>, #1, #3, R2 ; Set Z-bit. Clear N- and V-bits.
50 50 04 01 EF 0237 757 EXTZV #1, #4, R0, R0 ; Convert digit count to byte count
023C 758 BEQL 30$ ; Skip loop if zero or one digit
023E 759
023E 760 MARK_POINT MOVP_ACCVIO
83 81 90 023E 761 10$: MOVB (R1)+, (R3)+ ; Move next two digits
03 13 0241 762 BEQL 20$ ; Leave Z-bit alone if both zero
52 04 8A 0243 763 BICB #PSLSM_Z, R2 ; Otherwise, clear saved Z-bit
F5 50 F5 0246 764 20$: SOBGR R0, 10$ ; Check for end of loop
0249 765
0249 766 ; The last byte must be processed in a special way. The digit must be checked
0249 767 ; for nonzero because that affects the condition codes. The sign must be
0249 768 ; transformed into the preferred form. The N-bit must be set if the input
0249 769 ; is negative, but cleared in the case of negative zero.
0249 770
0249 771 MARK_POINT MOVP_ACCVIO
50 50 61 90 0249 772 30$: MOVB (R1), R0 ; Get last input byte (R1 now scratch)
50 F0 8F 93 024C 773 BITB #^B11110000, R0 ; Is digit nonzero?
03 13 0250 774 BEQL 40$ ; Branch if zero
52 04 8A 0252 775 BICB #PSLSM_Z, R2 ; Otherwise, clear saved Z-bit
51 50 F0 8F 8B 0255 776 40$: BICB3 #^B11110000, R0, R1 ; Sign "digit" to R1
025A 777
025A 778 ; Assume that the sign is "+". If the input sign is minus, one of the several
025A 779 ; fixups that must be done is to change the output sign from "+" to "-".
025A 780
50 04 00 0C F0 025A 781 INSV #12, #0, #4, R0 ; 12 is preferred plus sign
025F 782 CASE R1, LIMIT=#10, TYPE=B, <- ; Dispatch on sign type
025F 783 60$, - ; 10 => +
025F 784 50$, - ; 11 => -
025F 785 60$, - ; 12 => +
025F 786 50$, - ; 13 => -
025F 787 60$, - ; 14 => +
025F 788 60$, - ; 15 => +
025F 789 >
026F 790
026F 791 ; Input sign is "-"
026F 792
05 52 02 E0 026F 793 50$: BBS #PSLSV_Z, R2, 60$ ; Treat as "+" if negative zero
50 50 D6 0273 794 INCL R0 ; 13 is preferred minus sign
52 08 88 0275 795 BISB #PSLSM_N, R2 ; Set N-bit
0278 796
0278 797 ; Input sign is "+" or input is negative zero. Nothing special to do.
0278 798
0278 799 MARK_POINT MOVP_ACCVIO
63 50 90 0278 800 60$: MOVB R0, (R3) ; Move modified final digit
06 6E D4 027B 801 CLRL (SP) ; R0 and R2 must be zero on output
08 AE D4 027D 802 CLRL 8(SP) ; so clear saved R0 and R2
0F B9 0280 803 BICPSW #<PSLSM_N!PSLSM_Z!PSLSM_V!PSLSM_C> ; Clear all codes
52 B8 0282 804 BISPSW R2 ; Reset codes as appropriate
040F 8F BA 0284 805 POPR #^M<R0,R1,R2,R3,R10> ; Restore saved registers
05 0288 806 RSB ; Return
```



```
0289 808 .SUBTITLE Routine to Strip Leading Zeros from Decimal String
0289 809
0289 810 :+ Functional Description:
0289 811
0289 812 This routine strips leading (high-order) zeros from a packed decimal
0289 813 string. The routine exists based on two assumptions.
0289 814
0289 815 1. Many of the decimal strings that are used in packed decimal
0289 816 operations have several leading zeros.
0289 817
0289 818 2. The operations that are performed on a byte containing packed
0289 819 decimal digits are more complicated than the combination of this
0289 820 routine and any special end processing that occurs in the various
0289 821 VAX$xxxxxx routines when a string is exhausted.
0289 822
0289 823 This routine exists as a performance enhancement. As such, it can only
0289 824 succeed if it is extremely efficient. It does not attempt to be
0289 825 rigorous in squeezing every last zero out of a string. It eliminates
0289 826 only entire bytes that contain two zero digits. It does not look for a
0289 827 leading zero in the high order nibble of a string of odd length.
0289 828
0289 829 The routine also assumes that the input decimal strings are well
0289 830 formed. If an even-length decimal string does not have a zero in its
0289 831 unused high order nibble, then no stripping takes place, even though
0289 832 the underlying VAX$xxxxxx routine will work correctly.
0289 833
0289 834 (The comment in the next four lines is preserved for its historical
0289 835 content.)
0289 836
0289 837 Finally, there is no explicit test for the end of the string. The
0289 838 routine assumes that the low order byte, the one that contains the
0289 839 sign, is not equal to zero. This can cause rather strange behavior
0289 840 (read UNPREDICTABLE) for poorly formed decimal strings.
0289 841
0289 842 (The following comment describes the revised treatment of certain forms
0289 843 of illegal packed decimal strings.)
0289 844
0289 845 Although an end-of-string test is not required for well formed packed
0289 846 decimal strings, it turns out that some layered products create packed
0289 847 decimal data on the fly consisting of so many bytes containing zero. In
0289 848 other words, the sign nibble contains zero. Various implementations of
0289 849 the VAX architecture have treated these strings as representations of
0289 850 packed decimal zero.
0289 851
0289 852 The BLEQ 30$ instructions that exist in the following two loops detect
0289 853 these strings and treat them as strings with a digit count of one.
0289 854 (The digit itself is zero.) Whether this string is treated as +0 or -0
0289 855 is determined by the caller of this routine. That much UNPREDICTABLE
0289 856 behavior remains in the treatment of these illegal strings.
0289 857
0289 858 (End of revised comment)
0289 859
0289 860 : Input and Output Parameters:
0289 861
0289 862 There are really two identical but separate routines here. One is
0289 863 used when the input decimal string descriptor is in R0 and R1. The
0289 864 other is used when R2 and R3 describe the decimal string. Note that
```



```
0289 865 : we have already performed the reserved operand checks so that R0 (or
0289 866 : R2) is guaranteed LEQU 31.
0289 867 :
0289 868 : If the high order digit of an initially even length string is zero,
0289 869 : then the digit count (R0 or R2) is reduced by one. For all other
0289 870 : cases, the digit count is reduced by two as an entire byte of zeros
0289 871 : is skipped.
0289 872 :
0289 873 : Input Parameters (for entry at DECIMAL$STRIP_ZEROS_R0_R1):
0289 874 :
0289 875 : R0<4:0> - len.rw      Length of input decimal string
0289 876 : R1      - addr.ab      Address of input packed decimal string
0289 877 :
0289 878 : Output Parameters (for entry at DECIMAL$STRIP_ZEROS_R0_R1):
0289 879 :
0289 880 : R1      Advanced to first nonzero byte in string
0289 881 : R0      Reduced accordingly (Note that if R0 is altered at all,
0289 882 :         then R0 is always ODD on exit.)
0289 883 :
0289 884 : Input Parameters (for entry at DECIMAL$STRIP_ZEROS_R2_R3):
0289 885 :
0289 886 : R2<4:0> - len.rw      Length of input decimal string
0289 887 : R3      - addr.ab      Address of input packed decimal string
0289 888 :
0289 889 : Output Parameters (for entry at DECIMAL$STRIP_ZEROS_R2_R3):
0289 890 :
0289 891 : R3      Advanced to first nonzero byte in string
0289 892 : R2      Reduced accordingly (Note that if R2 is altered at all,
0289 893 :         then R2 is always ODD on exit.)
0289 894 :
0289 895 : Note:
0289 896 :
0289 897 : Although these routines can generate access violations, there is no
0289 898 : MARK POINT here because these routines can be called from other
0289 899 : modules (and are not called by the routines in this module). The PC
0289 900 : check is made based on the return PC from this subroutine rather than
0289 901 : on the PC of the instruction that accessed the inaccessible address.
0289 902 :-
0289 903 :
0289 904 : This routine is used when the decimal string is described by R0 (digit
0289 905 : count) and R1 (string address).
0289 906 :
0289 907 DECIMAL$STRIP_ZEROS_R0_R1::
06 50 E8 0289 908 BLBS R0,T0$ ; Skip first check if R0 starts out ODD
81 95 028C 909 TSTB (R1)+ ; Is first byte zero?
0D 12 028E 910 BNEQ 20$ ; All done if not
50 D7 0290 911 DECL R0 ; Skip leading zero digit (R0 NEQU 0)
81 95 0292 912
50 07 12 0294 913 10$: TSTB (R1)+ ; Is next byte zero?
02 C2 0296 914 BNEQ 20$ ; All done if not
05 15 0299 915 SUBL #2,R0 ; Decrease digit count by 2
F5 11 029B 916 BLEQ 30$ ; We passed the end of the string
51 D7 029D 917 BRB 10$ ; ... and charge on
05 05 029D 918
02A0 919 20$: DECL R1 ; Back up R1 to last nonzero byte
029F 920 RSB
02A0 921
```



```
50 02 C0 02A0 922 30$: ADDL #2,R0 ; Undo last R0 modification
    F8 11 02A3 923 BRB 20$ ; ... and take common exit
        02A5 924
        02A5 925 ; This routine is used when the decimal string is described by R2 (digit
        02A5 926 ; count) and R3 (string address).
        02A5 927
        06 52 E8 02A5 928 DECIMAL$STRIP_ZEROS_R2_R3::
            83 95 02A5 929 BLBS R2,T0$ ; Skip first check if R2 starts out ODD
            0D 12 02A8 930 TSTB (R3)+ ; Is first byte zero?
            52 D7 02AA 931 BNEQ 20$ ; All done if not
            02AE 932 DECL R2 ; Skip leading zero digit (R2 NEQU 0)
            83 95 02AE 933
            07 12 02B0 934 10$: TSTB (R3)+ ; Is next byte zero?
            52 02 C2 02B2 935 BNEQ 20$ ; All done if not
            05 15 02B5 936 SUBL #2,R2 ; Decrease digit count by 2
            F5 11 02B7 937 BLEQ 30$ ; We passed the end of the string
            02B9 938 BRB 10$ ; ... and charge on
            53 D7 02B9 939
            05 02BB 940 20$: DECL R3 ; Back up R3 to last nonzero byte
            02BC 941 RSB
            52 02 C0 02BC 942
            F8 11 02BF 943 30$: ADDL #2,R2 ; Undo last R2 modification
            02C1 944 BRB 20$ ; ... and take common exit
            945
```



```
02C1 947 .SUBTITLE DECIMAL_ROPRAND
02C1 948 :-
02C1 949 Functional Description:
02C1 950 :
02C1 951 This routine receives control when a digit count larger than 31
02C1 952 is detected. The exception is architecturally defined as an
02C1 953 abort so there is no need to store intermediate state. Because
02C1 954 all of the routines in this module check for legal digit counts
02C1 955 before saving any registers, this routine simply passes control
02C1 956 to VAX$ROPRAND.
02C1 957 :
02C1 958 Input Parameters:
02C1 959 :
02C1 960 0(SP) - Return PC from VAX$xxxxxx routine
02C1 961 :
02C1 962 Output Parameters:
02C1 963 :
02C1 964 0(SP) - Offset in packed register array to delta PC byte
02C1 965 4(SP) - Return PC from VAX$xxxxxx routine
02C1 966 :
02C1 967 Implicit Output:
02C1 968 :
02C1 969 This routine passes control to VAX$ROPRAND where further
02C1 970 exception processing takes place.
02C1 971 :-
02C1 972 :
02C1 973 ASSUME CMPP3_B_DELTA_PC EQ MOVP_B_DELTA_PC
02C1 974 ASSUME CMPP4_B_DELTA_PC EQ MOVP_B_DELTA_PC
02C1 975 :
02C1 976 DECIMAL_ROPRAND:
02C1 977 PUSHL #MOVP_B_DELTA_PC ; Store offset to delta PC byte
03 DD 02C1 977
FD3A' 31 02C3 978 BRW VAX$ROPRAND ; Pass control along
```



```
02C6 980 .SUBTITLE DECIMAL_ACCVIO - Reflect an Access Violation
02C6 981 ;+
02C6 982 ; Functional Description:
02C6 983 ;
02C6 984 ; This routine receives control when an access violation occurs while
02C6 985 ; executing within the emulator routines for CMPP3, CMPP4 or MOVP.
02C6 986 ;
02C6 987 ; The routine header for ASHP_ACCVIO in module VAX$ASHP contains a
02C6 988 ; detailed description of access violation handling for the decimal
02C6 989 ; string instructions.
02C6 990 ;
02C6 991 ; Input Parameters:
02C6 992 ;
02C6 993 ; See routine ASHP_ACCVIO in module VAX$ASHP
02C6 994 ;
02C6 995 ; Output Parameters:
02C6 996 ;
02C6 997 ; See routine ASHP_ACCVIO in module VAX$ASHP
02C6 998 ; -
02C6 999 ;
02C6 1000 DECIMAL_ACCVIO:
02C6 1001 CLRL R2 ; Initialize the counter
FD34 52 D4 02C8 1002 PUSHAB MODULE_BASE ; Store base address of this module
51 8E C2 02CC 1003 SUBL2 (SP)+,R1 ; Get PC relative to this base
02CF 1004
0000'CF42 51 B1 02CF 1005 10$: CMPW R1,PC_TABLE_BASE[R2] ; Is this the right PC?
07 13 02D5 1006 BEQL 30$ ; Exit loop if true
F4 52 0E F2 02D7 1007 AOBLS #TABLE_SIZE,R2,10$ ; Do the entire table
02DB 1008
02DB 1009 ; If we drop through the dispatching based on PC, then the exception is not
02DB 1010 ; one that we want to back up. We simply reflect the exception to the user.
02DB 1011
0F BA 02DB 1012 20$: POPR #^M<R0,R1,R2,R3> ; Restore saved registers
05 05 02DD 1013 RSB ; Return to exception dispatcher
02DE 1014
02DE 1015 ; The exception PC matched one of the entries in our PC table. R2 contains
02DE 1016 ; the index into both the PC table and the handler table. R1 has served
02DE 1017 ; its purpose and can be used as a scratch register.
02DE 1018
51 0000'CF42 3C 02DE 1019 30$: MOVZWL HANDLER_TABLE_BASE[R2],R1 ; Get the offset to the handler
FD17 CF41 17 02E4 1020 JMP MODULE_BASE[R1] ; Pass control to the handler
02E9 1021
02E9 1022 ; In all of the instruction-specific routines, the state of the stack
02E9 1023 ; will be shown as it was when the exception occurred. All offsets will
02E9 1024 ; be pictured relative to R0.
```



```
02E9 1026 .SUBTITLE Context-Specific Access Violation Handling for VAX$CMPPx
02E9 1027 :+
02E9 1028 : Functional Description:
02E9 1029 :
02E9 1030 : It is trivial to back out CMPP3 and CMPP4 because neither of these
02E9 1031 : routines uses any stack space (other than saved register space). The
02E9 1032 : only reason that this routine does not use the common
02E9 1033 : VAX$DECIMAL_ACCVIO exit path is that fewer registers are saved by
02E9 1034 : these two routines than are saved by the typical packed decimal
02E9 1035 : emulation routine.
02E9 1036 :
02E9 1037 : Input Parameters:
02E9 1038 :
02E9 1039 : R0 - Address of top of stack when access violation occurred
02E9 1040 :
02E9 1041 : 00(R0) - Saved R0 on entry to VAX$CMPPx
02E9 1042 : 04(R0) - Saved R1
02E9 1043 : 08(R0) - Saved R2
02E9 1044 : 12(R0) - Saved R3
02E9 1045 : 16(R0) - Saved R4
02E9 1046 : 20(R0) - Saved R5
02E9 1047 : 24(R0) - Saved R10
02E9 1048 : 28(R0) - Return PC from VAX$CMPPx routine
02E9 1049 :
02E9 1050 : 00(SP) - Saved R0 (restored by VAX$HANDLER)
02E9 1051 : 04(SP) - Saved R1
02E9 1052 : 08(SP) - Saved R2
02E9 1053 : 12(SP) - Saved R3
02E9 1054 :
02E9 1055 : Output Parameters:
02E9 1056 :
02E9 1057 : R0 is advanced over saved register array as the registers are restored.
02E9 1058 : R0 ends up pointing at the return PC.
02E9 1059 :
02E9 1060 : R1 contains the value of delta PC for all of the routines that
02E9 1061 : use this common code path. The FPD and ACCVIO bits are both set
02E9 1062 : in R1.
02E9 1063 :
02E9 1064 : 00(R0) - Return PC from VAX$CMPPx routine
02E9 1065 :
02E9 1066 : 00(SP) - Value of R0 on entry to VAX$CMPPx
02E9 1067 : 04(SP) - Value of R1 on entry to VAX$CMPPx
02E9 1068 : 08(SP) - Value of R2 on entry to VAX$CMPPx
02E9 1069 : 12(SP) - Value of R3 on entry to VAX$CMPPx
02E9 1070 :
02E9 1071 : R4, R5, and R10 are restored to their values on entry to VAX$CMPPx.
02E9 1072 : -
02E9 1073 :
02E9 1074 : .ENABLE LOCAL_BLOCK
02E9 1075 :
02E9 1076 CMPPx_ACCVIO::
02E9 1077 MOVQ (R0)+,PACK_L_SAVED_R0(SP) ; 'Restore' R0 and R1
02E9 1078 MOVQ (R0)+,PACK_L_SAVED_R2(SP) ; 'Restore' R2 and R3
02E9 1079 MOVQ (R0)+,R4 ; Really restore R4 and R5
02F3 1080
02F3 1081 ; The last two instructions can be shared with MOVP_ACCVIO, provided that
02F3 1082 ; the following assumptions hold.
```


VAX\$DECIMAL
V04-000

- VAX-11 Packed Decimal Instruction Emul 16-SEP-1984 01:30:59 VAX/VMS Macro V04-00 Page 24
Context-Specific Access Violation Handl 5-SEP-1984 00:45:12 [EMULAT.SRC]VAXDECIML.MAR;1 (12)

		02F3	1083		
		02F3	1084	ASSUME	CMPP3_B_DELTA_PC EQ MOV_P_B_DELTA_PC
		02F3	1085	ASSUME	CMPP4_B_DELTA_PC EQ MOV_P_B_DELTA_PC
		02F3	1086		
0D	11	02F3	1087	BRB	10\$; Share remainder with MOV_P_ACCVIO

VAX
V04

```
02F5 1089 .SUBTITLE Context-Specific Access Violation Handling for VAX$MOVP
02F5 1090 :+
02F5 1091 : Functional Description:
02F5 1092 :
02F5 1093 : It is almost too trivial to back out VAX$MOVP to its starting point.
02F5 1094 : If time permits, we will add restart points to this routine. This will
02F5 1095 : illustrate how one could go about adding restart capability to other
02F5 1096 : decimal instructions, allowing the routines to pick up where they left
02F5 1097 : off if an access violation occurs. This will also point out the
02F5 1098 : magnitude of the task by showing the amount of intermediate state that
02F5 1099 : must be saved for even so simple a routine as VAX$MOVP.
02F5 1100 :
02F5 1101 : The VAX$MOVP routine, like VAX$CMPPx, uses no stack space. It also
02F5 1102 : saves only a subset of the registers and so a special exit path must
02F5 1103 : be taken to VAX$REFLECT_FAULT.
02F5 1104 :
02F5 1105 : Input Parameters:
02F5 1106 :
02F5 1107 : R0 - Address of top of stack when access violation occurred
02F5 1108 :
02F5 1109 : 00(R0) - Saved R0 on entry to VAX$MOVP
02F5 1110 : 04(R0) - Saved R1
02F5 1111 : 08(R0) - Saved R2
02F5 1112 : 12(R0) - Saved R3
02F5 1113 : 16(R0) - Saved R10
02F5 1114 : 20(R0) - Return PC from VAX$MOVP routine
02F5 1115 :
02F5 1116 : 00(SP) - Saved R0 (restored by VAX$HANDLER)
02F5 1117 : 04(SP) - Saved R1
02F5 1118 : 08(SP) - Saved R2
02F5 1119 : 12(SP) - Saved R3
02F5 1120 :
02F5 1121 : Output Parameters:
02F5 1122 :
02F5 1123 : R0 is advanced over saved register array as the registers are restored.
02F5 1124 : R0 ends up pointing at the return PC.
02F5 1125 :
02F5 1126 : R1 contains the value of delta PC for all of the routines that
02F5 1127 : use this common code path. The FPD and ACCVIO bits are both set
02F5 1128 : in R1.
02F5 1129 :
02F5 1130 : 00(R0) - Return PC from VAX$MOVP routine
02F5 1131 :
02F5 1132 : 00(SP) - Value of R0 on entry to VAX$MOVP
02F5 1133 : 04(SP) - Value of R1 on entry to VAX$MOVP
02F5 1134 : 08(SP) - Value of R2 on entry to VAX$MOVP
02F5 1135 : 12(SP) - Value of R3 on entry to VAX$MOVP
02F5 1136 :
02F5 1137 : R10 is restored to its value on entry to VAX$MOVP.
02F5 1138 :
02F5 1139 :-
02F5 1140 MOV_P_ACCVIO::
02F5 1141 MOVQ (R0)+,PACK_L_SAVED_R0(SP) ; 'Restore' R0 and R1
02F5 1142 MOVQ (R0)+,PACK_L_SAVED_R2(SP) ; 'Restore' R2 and R3
02F5 1143 BISB3 #MOV_P_M_FPD,-8(R0),- ;
02F5 1144 MOV_P_B_STATE(SP) ; Preserve saved C-bit
0300 1144
0302 1145
```

```
6E 80 7D
08 AE 80 7D
FB A0 10 89
      02 AE
```



```
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
```

VAX\$DECIMAL
Symbol table

J 7

- VAX-11 Packed Decimal Instruction Emul 16-SEP-1984 01:30:59 VAX/VMS Macro V04-00 Page 27
5-SEP-1984 00:45:12 [EMULAT.SRC]VAXDECIML.MAR;1 (13)

...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

VAX
V04

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

PSECT name	Allocation	PSECT No.	Attributes
ABS	00000000 (0.)	00 (0.)	NOPIC USR CON ABS LCL NOSHR NOEXE NORD NOWRT NOVEC BYTE
\$ABSS	00000000 (0.)	01 (1.)	NOPIC USR CON ABS LCL NOSHR EXE RD WRT NOVEC BYTE
VAX\$CODE	0000030F (78.)	02 (2.)	PIC USR CON REL LCL SHR EXE RD NOWRT NOVEC LONG
PC TABLE	0000001C (2.)	03 (3.)	PIC USR CON REL LCL SHR NOEXE RD NOWRT NOVEC BYTE
HANDLER_TABLE	0000001C (2.)	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
\$255\$DUA28:[EMULAT.OBJ]VAXMACROS.MLB;1	10
\$255\$DUA28:[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

