
SSSSSSSSSSS DDDDDDDDDDD AAAAAAAAA
SSSSSSSSSSS DDDDDDDDDDD AAAAAAAAA
SSSSSSSSSSS DDDDDDDDDDD AAAAAAAAA
SSS DDD DDD AAA AAA
SSS DDD DDD AAA AAA
SSS DDD DDD AAA AAA
SSS DDD DDD AAA AAA
SSS DDD DDD AAA AAA
SSS DDD DDD AAA AAA
SSSSSSSSS DDD DDD AAA AAA
SSSSSSSSS DDD DDD AAA AAA
SSSSSSSSS DDD DDD AAA AAA
SSS DDD DDD AAAAAAAAAAAAAAA
SSS DDD DDD AAAAAAAAAAAAAAA
SSS DDD DDD AAAAAAAAAAAAAAA
SSS DDD DDD AAA AAA
SSS DDD DDD AAA AAA
SSS DDD DDD AAA AAA
SSSSSSSSSSS DDDDDDDDDDD AAA AAA
SSSSSSSSSSS DDDDDDDDDDD AAA AAA
SSSSSSSSSSS DDDDDDDDDDD AAA AAA

```
DDDDDDDD    EEEEEEEEEE    CCCCCCCC    OOOOOO    DDDDDDDD    EEEEEEEEEE
DDDDDDDD    EEEEEEEEEE    CCCCCCCC    OOOOOO    DDDDDDDD    EEEEEEEEEE
DD      DD  EE            CC          OO    OO  DD      DD  EE
DD      DD  EE            CC          OO    OO  DD      DD  EE
DD      DD  EE            CC          OO    OO  DD      DD  EE
DD      DD  EE            CC          OO    OO  DD      DD  EE
DD      DD  EEEEEEEE      CC          OO    OO  DD      DD  EEEEEEEE
DD      DD  EEEEEEEE      CC          OO    OO  DD      DD  EEEEEEEE
DD      DD  EE            CC          OO    OO  DD      DD  EE
DD      DD  EE            CC          OO    OO  DD      DD  EE
DD      DD  EE            CC          OO    OO  DD      DD  EE              ....
DDDDDDDD    EEEEEEEEEE    CCCCCCCC    OOOOOO    DDDDDDDD    EEEEEEEEEE      ....
DDDDDDDD    EEEEEEEEEE    CCCCCCCC    OOOOOO    DDDDDDDD    EEEEEEEEEE      ....

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
```

```
;     1          0001  0 %TITLE 'Instruction decoder'
;     2          0002  0 MODULE lib$ins_decode (IDENT = 'V04-000',
;     3          0003  0         ADDRESSING_MODE (EXTERNAL = LONG_RELATIVE)) =
;     4          0004  1 BEGIN
;     5          0005  1
;     6          0006  1 !++
;     7          0007  1 !
;     8          0008  1 !*****************************************************************************
;     9          0009  1 !*                                                                           *
;    10          0010  1 !*  COPYRIGHT (c) 1978, 1980, 1982, 1984 BY                                 *
;    11          0011  1 !*  DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.                   *
;    12          0012  1 !*  ALL RIGHTS RFSERVED.                                                     *
;    13          0013  1 !*                                                                           *
;    14          0014  1 !*  THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED   *
;    15          0015  1 !*  ONLY IN  ACCORDANCE WITH  THE  TERMS  OF  SUCH  LICENSE  AND WITH THE  *
;    16          0016  1 !*  INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR  ANY  OTHER   *
;    17          0017  1 !*  COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY   *
;    18          0018  1 !*  OTHER PERSON.  NO TITLE TO AND OWNERSHIP OF  THE  SOFTWARE IS  HEREBY   *
;    19          0019  1 !*  TRANSFERRED.                                                             *
;    20          0020  1 !*                                                                           *
;    21          0021  1 !*  THE INFORMATION IN THIS SOFTWARE IS  SUBJECT TO CHANGE WITHOUT NOTICE   *
;    22          0022  1 !*  AND  SHOULD  NOT  BE  CONSTRUED AS  A COMMITMENT BY DIGITAL EQUIPMEN'   *
;    23          0023  1 !*  CORPORATION.                                                             *
;    24          0024  1 !*                                                                           *
;    25          0025  1 !*  DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE  OR  RELIABILITY OF ITS   *
;    26          0026  1 !*  SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.                  *
;    27          0027  1 !*                                                                           *
;    28          0028  1 !*                                                                           *
;    29          0029  1 !*****************************************************************************
;    30          0030  1 !
;    31          0031  1 ! FACILITY:
;    32          0032  1 !       VAX instruction decoder.
;    33          0033  1 !
;    34          0034  1 ! Portions taken from DBGINS module by KEVIN PAMMETT, 2-MAR-77
;    35          0035  1 !
;    36          0036  1 ! Author:       Tim Halvorsen, 09-Feb-1981
;    37          0037  1 !
;    38          0038  1 ! Modified by:
;    39          0039  1 !
;    40          0040  1 !       V002    TMH0002         Tim Halvorsen   09-Aug-1981
;    41          0041  1 !               Remove SHR psect attribute so linker doesn't generate a
;    42          0042  1 !               non-crf writable section, and the imgact doesn't try to map
;    43          0043  1 !               a read/write shared section to the .EXE file.
;    44          0044  1 !
;    45          0045  1 !       V001    TMH0001         Tim Halvorsen   09-Mar-1981
;    46          0046  1 !               Use PLIT psect rather than OWN psect for read-only
;    47          0047  1 !               data arrays.  Make each failure status a separate
;    48          0048  1 !               code to aid in debugging the case of a decode failure.
;    49          0049  1 !               Remove probes of instruction stream because a PROBER
;    50          0050  1 !               instruction determines the access from the previous
;    51          0051  1 !               mode, not the current mode.  Thus, if you call this
;    52          0052  1 !               routine with a stream readable only to the current mode,
;    53          0053  1 !               it will fail.  For now, we skip the checks and allow
;    54          0054  1 !               an access violation to occur within the routine.
;    55          0055  1 !--
;    56          0056  1
;    57          0057  1 !
```

```
;    58          0058  1 ! Require and Library files:
;    59          0059  1 !
;    60          0060  1
;    61          0061  1 LIBRARY 'SYS$LIBRARY:STARLET';          ! Standard VMS definitions
;    62          0062  1 SWITCHES LIST(REQUIRE);
;    63          0063  1 REQUIRE 'SRC$:VAXOPS';                  ! Literals and macros related to opcodes
```

```
;  VAXOPS.REQ - OP CODE TABLE FOR VAX INSTRUCTIONS
;
; Version:      'V04-000'
;
;****************************************************************************
;*                                                                          *
;*  COPYRIGHT (c) 1978, 1980, 1982, 1984 BY                                 *
;*  DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.                  *
;*  ALL RIGHTS RESERVED.                                                    *
;*                                                                          *
;*  THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED   *
;*  ONLY IN  ACCORDANCE WITH  THE  TERMS  OF  SUCH  LICENSE  AND WITH THE   *
;*  INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR  ANY  OTHER   *
;*  COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY   *
;*  OTHER PERSON.  NO TITLE TO AND OWNERSHIP OF  THE  SOFTWARE IS  HEREBY   *
;*  TRANSFERRED.                                                            *
;*                                                                          *
;*  THE INFORMATION IN THIS SOFTWARE IS  SUBJECT TO CHANGE WITHOUT NOTICE   *
;*  AND  SHOULD  NOT  BE  CONSTRUED AS  A COMMITMENT BY DIGITAL EQUIPMENT   *
;*  CORPORATION.                                                            *
;*                                                                          *
;*  DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE  OR  RELIABILITY OF ITS   *
;*  SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.                 *
;*                                                                          *
;*                                                                          *
;****************************************************************************
;
; Author:
;               KEVIN PAMMETT, MARCH 2, 1977.
;
; Modified by:
;
;       V001    TMH0001         Tim Halvorsen   09-Feb-1981
;               Rewrite macro invocations to supply the entire SRM
;               operand specification, to allow checking for literals
;               in write operands, and other invalid conditions.
;--

LITERAL

;
; OPERAND ACCESS TYPE (A,B,M,R,V,W) - 1 BIT WIDE
;

        ACCESS_A = 0,                   ; EFFECTIVE ADDRESS
        ACCESS_B = 0,                   ; BRANCH DISPLACEMENT
        ACCESS_R = 1,                   ; OPERAND IS READ-ONLY
        ACCESS_W = 0,                   ; OPERAND IS WRITE-ONLY
        ACCESS_M = 0,                   ; OPERAND IS MODIFIED
        ACCESS_V = 0,                   ; ADDRESS A SET OF 2 REGISTERS

;
; OPERAND DATA TYPE (B,W,L,Q,F,D,G,H,V) - 3 BITS WIDE
;

        DATA_B = 0,                     ; BYTE CONTEXT
        DATA_W = 1,                     ; WORD CONTEXT
```

```
;  R0121  1              DATA_L = 2,                     ! LONGWORD CONTEXT
;  R0122  1              DATA_Q = 3,                     ! QUADWORD CONTEXT
;  R0123  1              DATA_F = DATA_L,                ! FLOATING CONTEXT
;  R0124  1              DATA_D = DATA_Q,                ! FLOATING DOUBLE CONTEXT (8 BYTES)
;  R0125  1              DATA_G = DATA_Q,                ! FLOATING GRAND CONTEXT (8 BYTES)
;  R0126  1              DATA_H = 4,                     ! FLOATING HUGE CONTEXT (16 BYTES)
;  R0127  1
;  R0128  1      !
;  R0129  1      ! BRANCH DISPLACEMENT TYPES
;  R0130  1      !
;  R0131  1
;  R0132  1              NO_BRANCH = 0,                  ! NO BRANCH
;  R0133  1              BRANCH_BYTE = 1,                ! BRANCH BYTE
;  R0134  1              BRANCH_WORD = 2;                ! BRANCH WORD
;  R0135  1
;  R0136  1      !
;  R0137  1      ! THE FOLLOWING MACRO IS USED TO BUILD SUCCESSIVE ENTRIES FOR
;  R0138  1      ! THE TABLE.  EACH MACRO CALL CONTAINS THE
;  R0139  1      ! INFO FOR 1 VAX OPCODE, AND THE ENTRIES ARE SIMPLY
;  R0140  1      ! BUILT IN THE ORDER THAT THE MACRO CALLS ARE MADE -
;  R0141  1      ! THE ASSUMPTION IS THAT THEY WILL BE MADE IN ORDER OF
;  R0142  1      ! INCREASING OPCODE VALUES.  THIS IS NECESSARY BECAUSE
;  R0143  1      ! THE TABLE IS ACCESSED BY USING A GIVEN OPCODE AS THE
;  R0144  1      ! TABLE INDEX.
;  R0145  1      !
;  R0146  1
;  R0147  1      COMPILETIME $BRANCH_TYPE=0;
;  R0148  1
;  R0149  1      MACRO
;  R0150  1          GET_1ST(A,B) = A%,
;  R0151  1          GET_2ND(A,B) = B%,
; MR0152  1          OPERAND(NAME) =
; MR0153  1              %IF %NULL(NAME)
; MR0154  1              %THEN
; MR0155  1                  0
; MR0156  1              %ELSE
; MR0157  1                  BEGIN
; MR0158  1      !           %IF NOT %DECLARED(%STRING('ACCESS_',GET_1ST(%EXPLODE(NAME))))
; MR0159  1      !           %THEN
; MR0160  1      !               %WARN('Invalid access type ',GET_1ST(%EXPLODE(NAME)))
; MR0161  1      !           %FI
; MR0162  1      !           %IF NOT %DECLARED(%STRING('DATA_',GET_2ND(%EXPLODE(NAME))))
; MR0163  1      !           %THEN
; MR0164  1      !               %WARN('Invalid data type ',GET_2ND(%EXPLODE(NAME)))
; MR0165  1      !           %FI
; MR0166  1                  %IF NAME EQL 'BB'
; MR0167  1                  %THEN
; MR0168  1                      %ASSIGN($BRANCH_TYPE, BRANCH_BYTE)
; MR0169  1                  %ELSE %IF NAME EQL 'BW'
; MR0170  1                  %THEN
; MR0171  1                      %ASSIGN($BRANCH_TYPE, BRANCH_WORD)
; MR0172  1                  %FI %FI
; MR0173  1                  %NAME('DATA_',GET_2ND(%EXPLODE(NAME))) +
; MR0174  1                      %NAME('ACCESS_',GET_1ST(%EXPLODE(NAME))) ^ 3
; MR0175  1                  END
;  R0176  1              %FI %,
;  R0177  1
```

```
;MR0178  1          OPDEF(NAME, OPC, OP1, OP2, OP3, OP4, OP5, OP6) =
;MR0179  1              %ASSIGN($BRANCH_TYPE,NO_BRANCH)
;MR0180  1              %RAD50_11 NAME,                         ! Opcode name in RAD50
;MR0181  1              %IF GET_1ST(%EXPLODE(NAME)) EQL 'X'     ! If undefined opcode,
;MR0182  1                  AND GET_2ND(%EXPLODE(NAME)) EQL 'X'
;MR0183  1              %THEN
;MR0184  1                  NOT_AN_OP                           ! then no operands
;MR0185  1              %ELSE
;MR0186  1                  %LENGTH-2                           ! else, number of operands
;MR0187  1              %FI OR
;MR0188  1                  OPERAND(OP1)^4,                     ! Define each operand
;MR0189  1              OPERAND(OP2) OR
;MR0190  1                  OPERAND(OP3)^4,
;MR0191  1              OPERAND(OP4) OR
;MR0192  1                  OPERAND(OP5)^4,
;MR0193  1              OPERAND(OP6) OR
 R0194  1                  $BRANCH_TYPE^4%;                    ! Define branch context
 R0195  1
 R0196  1      !
 R0197  1      ! MACROS TO ACCESS THE FIELDS.
 R0198  1      !
 R0199  1
 R0200  1      MACRO
 R0201  1          OP_NAME     = 0,0,32,0%,                  ! OPCODE MNEUMONIC (6 RAD50 CHARS)
 R0202  1          OP_NUMOPS   = 4,0,4,0%,                   ! NUMBER OF OPERANDS
 R0203  1          OP_CONTEXT(I) = 4+I/2, ((I) AND 1)*4, 3, 0 %, ! OPERAND CONTEXT
 R0204  1          OP_DATATYPE(I) = 4+I/2, ((I) AND 1)*4 + 3, 1, 0 %, ! OPERAND DATA TYPE
 R0205  1          OP_BR_TYPE = 7,4,4,0 %;                   ! CONTEXT OF BRANCH DISPLACEMENT
 R0206  1
 R0207  1      LITERAL
 R0208  1                  OPTSIZE = 8,                      ! EACH OPINFO BLOCK IS 9 BYTES LONG.
 R0209  1                  MAXOPCODE = %X'FD',               ! MAXIMUM VAX OP CODE WHICH IS VALID.
 R0210  1                  MAXOPRNDS = 6,                    ! MAXIMUM NUMBER OF OPERANDS PER INSTRUCTION.
 R0211  1                                                    ! NO INSTRUCTION THAT HAS BRANCH TYPE ADDRESSING
 R0212  1                                                    ! CAN HAVE THIS MANY OPERANDS UNLESS WE CHANGE
 R0213  1                                                    ! THE ORGANIZATION OF EACH OPINFO BLOCK.
 R0214  1                  BITS_PER_BYTE = 8,                ! NUMBER OF BITS IN A VAX BYTE.
 R0215  1                  AP_REG  = 12,                     ! NUMBER OF PROCESSOR REGISTER, 'AP'.
 R0216  1                  PC_REG  = 15,                     ! NUMBER OF PROCESSOR REGISTER, 'PC'.
 R0217  1
 R0218  1                  PC_REL_MODE = 8,                  ! ADDRESSING MODE: (PC)+
 R0219  1                  AT_PC_REL_MODE = 9,               ! ADDRESSING MODE: @(PC)+
 R0220  1                  INDEXING_MODE = 4,                ! ADDRESSING MODE: XXX[RX]
 R0221  1
 R0222  1                  SHORT_LIT_AMODE = 0,              ! Short literals fit right into the mode byte.
 R0223  1                  REGISTER_AMODE  = 5,              ! Register mode addressing.
 R0224  1                  REG_DEF_AMODE   = 6,              ! Register deferred addressing mode.
 R0225  1                  AUTO_DEC_AMODE  = 7,              ! Auto decrement addressing mode.
 R0226  1                  AUTO_INC_AMODE  = 8,              ! Auto Increment addressing mode.
 R0227  1                  DISP_BYTE_AMODE = 10,             ! All of the displacement modes start from
 R0228  1                                                    !  here.  See ENC_OPERAND() IN DBGENC.B32
 R0229  1                  DISP_LONG_AMODE = 14,
 R0230  1                  OP_CH_SIZE = 6;                   ! SIZE, IN ASCII CHARS, OF OPCODE MNEMONIC.
 R0231  1
 RU232  1      MACRO
 R0233  1                  DSPL_MODE = 0,4,4,0 %,            ! ADDRESSING MODE BITS FROM THE DOMINANT MODE
 R0234  1                                                    !  BYTE OF AN OPERAND REFERENCE.
```

```
;  R0235  1              DOM_MOD_FIELD = 0,5,2,1 %,
;  R0236  1                                      ! BITS WHICH WE PICK UP TO DIFFERENTIATE CERTAIN
;  R0237  1                                      !  TYPES OF DOMINANT MODES.  SEE DBGMAC.B32
;  R0238  1              SHORT_LITERAL = 0,0,6,0 %,
;  R0239  1                                      ! HOW TO EXCTRACT A 'SHORT LITERAL' FROM
;  R0240  1                                      !   THE INSTRUCTION STREAM.  SEE SRM.
;  R0241  1              AMODE    = 0,4,4,1 %,   ! BITS OF DOMINANT MODE ADDRESSING BYTE
;  R0242  1                                      !   WHICH SPECIFY THE ACTUAL MODE.
;  R0243  1              AREG     = 0,0,4,0 %,   ! BITS OF DOMINANT MODE ADDRESSING BYTE
;  R0244  1                                      !   WHICH SPECIFY REGISTER NUMBER, ETC.
;  R0245  1              NOT_AN_OP = 15 %,      ! OP_NUMOPS INDICATOR FOR UNASSIGNED OPCODES.
;  R0246  1              RESERVED = 'UNUSED' %; ! NAME OF RESERVED OPCODES.
;  R0247  1
;  R0248  1
;  R0249  1      MACRO
;  R0250  1              NEXT_FIELD(INDEX)      ! USED TO GET THE ADDRESS OF THE NEXT
;  R0251  1                                     !   FIELD OF A BLOCK.
;  R0252  1                      = (INDEX),0,0,0 %;
;  R0253  1
;  R0254  1      !  MACROS AND LITERALS SPECIFICALLY FOR INSTRUCTON ENCODING.
;  R0255  1      !  ('MACHINE -IN'.)
;  R0256  1
;  R0257  1      LITERAL
;  R0258  1              BAD_OPCODE       = 1,   ! CAN'T INTERPRET THE GIVEN ASCII OPCODE.
;  R0259  1              BAD_OPERAND      = 2,   ! UNDECODABLE OPERAND REFERENCE.
;  R0260  1              BAD_OPRNDS       = 3,   ! WRONG NUMBER OF OPERANDS.
;  R0261  1              INS_RESERVED     = 4;   ! GIVEN OPCODE IS RESERVED.
;  R0262  1
;  R0263  1      LITERAL
;  R0264  1                      ! We only have to special-case a few OPCODES,
;  R0265  1
;  R0266  1
;  R0267  1              OP_CASEB         = %X'8F',
;  R0268  1              OP_CASEW         = %X'AF',
;  R0269  1              OP_CASEL         = %X'CF';
;  R0270  1      !++
;  R0271  1      !
;  R0272  1      !       TOKEN VALUES USED FOR ENCODING/DECODING
;  R0273  1      !
;  R0274  1      !--
;  R0275  1
;  R0276  1      LITERAL
;  R0277  1              indexing_token   = 240,
;  R0278  1              val_token        = 241,
;  R0279  1              byte_val_token   = val_token + %SIZE(VECTOR[1,BYTE]),    ! 242
;  R0280  1              word_val_token   = val_token + %SIZE(VECTOR[1,WORD]),    ! 243
;  R0281  1              brch_token       = 244,
;  R0282  1              long_val_token   = val_token + %SIZE(VECTOR[1,LONG]),    ! 245
;  R0283  1              at_reg_token     = 246,
;  R0284  1              register_token   = 247,
;  R0285  1              lit_token        = 248,
;  R0286  1              bad_token        = 249;
;  R0287  1
;  R0288  1
;  R0289  1      ! The following structure declaration selects the proper opcode
;  R0290  1      ! table by looking for the extended opcode opcode(s).
;  R0291  1
```

```
;  R0292  1     STRUCTURE OPCODE_TBL [OPC,O,P,S,E] =
;  R0293  2             BEGIN
;  R0294  2             EXTERNAL LIB$GB_OPINFO1 : BLOCKVECTOR[256,OPTSIZE,BYTE];
;  R0295  2             EXTERNAL LIB$GB_OPINFO2 : BLOCKVECTOR[256,OPTSIZE,BYTE];
;  R0296  2             LOCAL OFFSET;
;  R0297  2             OFFSET = 0;
;  R0298  2             IF (OPC AND %X'FF') NEQ %X'FD'
;  R0299  2             THEN LIB$GB_OPINFO1[OPC,.OFFSET,0,8,0]  ! One byte opcodes
;  R0300  2             ELSE LIB$GB_OPINFO2[(OPC^-8),.OFFSET,0,8,0]     ! Two byte opcodes
;  R0301  1             END<P,S,E>;
;  R0302  1
;  R0303  1     !       VAXOPS.REQ      - last line
```

```
;    65          0304  1 %SBTTL 'Module declarations'
;    66          0305  1
;    67          0306  1 !
;    68          0307  1 ! Table of contents:
;    69          0308  1 !
;    70          0309  1
;    71          0310  1 LINKAGE
;    72          0311  1     ptr_linkage = CALL: GLOBAL(stream_ptr=11),
;    73          0312  1     append_linkage = JSB(REGISTER=0,REGISTER=1);
;    74          0313  1
;    75          0314  1 FORWARD ROUTINE
;    76          0315  1     lib$ins_decode,                         ! decode an instruction.
;    77          0316  1     ins_operand:        ptr_linkage,        ! print out an operand reference.
;    78          0317  1     branch_type:        ptr_linkage,        ! handle branch type addressing.
;    79          0318  1     displacement:       ptr_linkage,        ! extract displacement from instruction
;    80          0319  1     ins_context,                            ! get expected context of an operand
;    81          0320  1     put_reg:            NOVALUE,            ! print a register reference.
;    82          0321  1     append_address:     NOVALUE,            ! Append an address
;    83          0322  1     append_hex:         NOVALUE,            ! Append a hex value
;    84          0323  1     append_decimal:     NOVALUE,            ! Append an unsigned decimal value
;    85          0324  1     append_rad50:       NOVALUE,            ! Append a RAD50 string
;    86          0325  1     append_string:      append_linkage NOVALUE; ! Append string to the output buffer
;    87          0326  1
;    88          0327  1 !
;    89          0328  1 ! Psect declarations
;    90          0329  1 !
;    91          0330  1
;    92          0331  1 PSECT
;    93          0332  1     OWN  = z$debug_code(PIC,WRITE,EXECUTE,ALIGN(2)),
;    94          0333  1     CODE = z$debug_code(PIC,WRITE,EXECUTE,ALIGN(2)),
;    95          0334  1     PLIT = z$debug_code(PIC,WRITE,EXECUTE,ALIGN(2));
;    96          0335  1
;    97          0336  1 !
;    98          0337  1 ! Equated symbols:
;    99          0338  1 !
;   100          0339  1
;   101          0340  1 LITERAL
;   102          0341  1     true = 1,
;   103          0342  1     false = 0,
;   104          0343  1     round_brackets      = 0,                ! These are all flag parameters to
;   105          0344  1     square_brackets     = 2,                !    the routine 'PUT_REG'.
;   106          0345  1     no_brackets         = 1;
;   107          0346  1
;   108          0347  1 !
;   109          0348  1 ! OWN storage for up-level references
;   110          0349  1 !
;   111          0350  1
;   112          0351  1 OWN
;   113          0352  1     user_symbolize_routine,                 ! Address of user symbolize routine
;   114          0353  1     user_buffer_address,                    ! Address of user buffer storage
;   115          0354  1     user_buffer_size:   WORD,               ! Size of user buffer
;   116          0355  1     user_buffer_left:   WORD,               ! # bytes left in user buffer to fill
;   117          0356  1     last_literal_value;                     ! Value of last operand
;   118          0357  1
;   119          0358  1 !
;   120          0359  1 ! Macro to invoke a command, and return if the resultant value is an error
;   121          0360  1 !
```

```
;  122          0361  1
;  123          0362  1 MACRO
;  124        M 0363  1     return_if_error(command) =
;  125        M 0364  1         BEGIN
;  126        M 0365  1         LOCAL
;  127        M 0366  1             status;
;  128        M 0367  1
;  129        M 0368  1         status = command;
;  130        M 0369  1         IF NOT .status
;  131        M 0370  1         THEN
;  132        M 0371  1             RETURN .status;
;  133          0372  1         END%;
;  134          0373  1
;  135          0374  1 !
;  136          0375  1 ! Macro to probe read accessibility of a data segment
;  137          0376  1 !
;  138          0377  1
;  139          0378  1 MACRO
;  140        M 0379  1     probe(address,length) =
;  141        M 0380  1         BEGIN
;  142        M 0381  1 !       BUILTIN PROBER;
;  143        M 0382  1 !       IF NOT PROBER(%REF(0),%REF(length),address)
;  144        M 0383  1 !       THEN
;  145        M 0384  1 !           RETURN lib$_accvio;
;  146        M 0385  1         true
;  147          0386  1         END%;
;  148          0387  1
;  149          0388  1 !
;  150          0389  1 ! Macro to append a string to the output buffer
;  151          0390  1 !
;  152          0391  1
;  153          0392  1 MACRO
;  154        M 0393  1     append(string) =
;  155        M 0394  1         append_string(%CHARCOUNT(string),UPLIT BYTE(string)
;  156          0395  1         %IF %LENGTH GTR 1 %THEN ,%REMAINING %FI)%;
;  157          0396  1 !
;  158          0397  1 ! External storage
;  159          0398  1 !
;  160          0399  1
;  161          0400  1 EXTERNAL
;  162          0401  1     lib$gb_opinfo: opcode_tbl;          ! Table describing VAX instruction set
;  163          0402  1
;  164          0403  1 !
;  165          0404  1 ! Define message codes
;  166          0405  1 !
;  167          0406  1
;  168          0407  1 LITERAL
;  169          0408  1     lib$_accvio = 0,
;  170          0409  1     lib$_noinstran = 2,
;  171          0410  1     lib$_numtrunc = 4;
```

```
;  173          0411  1 GLOBAL ROUTINE lib$ins_decode(stream_pntr, outbuf, retlen, symbolize_rtn) =
;  174          0412  1
;  175          0413  1 !---
;  176          0414  1 !       This routine examines a byte stream that it is passed
;  177          0415  1 !       a pointer to, and tries to output what instructions
;  178          0416  1 !       this corresponds to symbolically.
;  179          0417  1 !
;  180          0418  1 ! Inputs:
;  181          0419  1 !
;  182          0420  1 !       stream_ptnr = Address of a byte pointer to the instruction stream.
;  183          0421  1 !       outbuf = Address of a buffer descriptor to receive the
;  184          0422  1 !                       decoded instruction
;  185          0423  1 !       symbolize_rtn = Address of a routine to call to convert an address
;  186          0424  1 !
;  187          0425  1 !
;  188          0426  1 ! Outputs:
;  189          0427  1 !
;  190          0428  1 !       R0 = status code
;  191          0429  1 !       The stream_pntr is updated to point to the next instruction.
;  192          0430  1 !--
;  193          0431  1
;  194          0432  2 BEGIN
;  195          0433  2
;  196          0434  2 BUILTIN
;  197          0435  2     NULLPARAMETER;
;  198          0436  2
;  199          0437  2 MAP
;  200          0438  2     stream_pntr: REF VECTOR [,LONG],
;  201          0439  2     outbuf:      REF BLOCK [,BYTE],
;  202          0440  2     retlen:      REF VECTOR [,WORD];
;  203          0441  2
;  204          0442  2 GLOBAL REGISTER
;  205          0443  2     stream_ptr=11: REF VECTOR[,BYTE];   ! Points to the instruction stream
;  206          0444  2
;  207          0445  2 LOCAL
;  208          0446  2     opcode: WORD;                       ! Instruction opcode
;  209          0447  2
;  210          0448  2 stream_ptr = .stream_pntr [0];          ! Get pointer to instruction stream
;  211          0449  2
;  212          0450  2 user_buffer_size = .outbuf [dsc$w_length];
;  213          0451  2 user_buffer_address = .outbuf [dsc$a_pointer];
;  214          0452  2 user_buffer_left = .user_buffer_size;
;  215          0453  2
;  216          0454  2 IF NULLPARAMETER(4)                     ! If 4th parameter unspecified,
;  217          0455  2 THEN
;  218          0456  2     user_symbolize_routine = 0          ! then set no routine
;  219          0457  2 ELSE
;  220          0458  2     user_symbolize_routine = .symbolize_rtn;
;  221          0459  2
;  222          0460  2 probe(.stream_ptr,1);                   ! Exit if we can't read the opcode
;  223          0461  2
;  224          0462  2 !
;  225          0463  2 !  Pick up the opcode and it check for validity.
;  226          0464  2 !
;  227          0465  2
;  228          0466  2 opcode = .stream_ptr [0];               ! Get first byte of opcode
;  229          0467  2
```

```
;  230          0468  2 IF .opcode EQL %X'FD'                       ! Check to see if 2 byte opcode
;  231          0469  2 THEN
;  232          0470  3     BEGIN                                     ! It is. Get the next byte of opcode.
;  233          0471  3     opcode = .stream_ptr [1]^8 + .opcode;
;  234          0472  3     stream_ptr = .stream_ptr + 1;
;  235          0473  2     END;
;  236          0474  2
;  237          0475  2 IF .opcode EQL %X'FF'                       ! If bugcheck opcode,
;  238          0476  2     AND .stream_ptr [1] EQL %X'FE'
;  239          0477  2 THEN
;  240          0478  3     BEGIN
;  241          0479  3     probe(.stream_ptr,4);                     ! Make sure all 4 bytes are readable
;  242          0480  3     append('BUG_CHECK #');
;  243          0481  3     append_hex(.(.stream_ptr+2)<0,16,0>,2);
;  244          0482  3     stream_pntr [0] = .stream_ptr+4;          ! Point to next instruction
;  245          0483  3     IF NOT NULLPARAMETER(3)                   ! If RETLEN specified,
;  246          0484  3     THEN
;  247          0485  3         retlen [0] = .user_buffer_size - .user_buffer_left;
;  248          0486  3     RETURN ss$_normal;
;  249          0487  2     END;
;  250          0488  2
;  251          0489  2 IF .lib$gb_opinfo[.opcode, op_numops] EQL not_an_op ! If unknown opcode,
;  252          0490  2 THEN
;  253          0491  2     RETURN lib$_noinstran;                    ! Unable to translate instruction
;  254          0492  2
;  255          0493  2 !
;  256          0494  2 ! Bump the instruction pointer up past the opcode,
;  257          0495  2 ! and output the character sequence which corresponds to it.
;  258          0496  2 !
;  259          0497  2
;  260          0498  2 stream_ptr = .stream_ptr + 1;
;  261          0499  2
;  262          0500  2 append_rad50(op_ch_size/3, lib$gb_opinfo [.opcode, op_name]);
;  263          0501  2 append('   ');
;  264          0502  2
;  265          0503  2 !
;  266          0504  2 ! Loop, encoding how each operand is referenced.
;  267          0505  2 !
;  268          0506  2
;  269          0507  2 INCR I FROM 1 TO .lib$gb_opinfo [.opcode, op_numops]
;  270          0508  2 DO
;  271          0509  3     BEGIN
;  272          0510  3     return_if_error(ins_operand(.i, .opcode));
;  273          0511  3
;  274          0512  3     IF .i NEQ 0 AND .i LSS .lib$gb_opinfo [.opcode, op_numops]
;  275          0513  3     THEN
;  276          0514  3         append(',');
;  277          0515  2     END;
;  278          0516  2
;  279          0517  2 !
;  280          0518  2 ! For CASE instructions, increment the stream pointer past the
;  281          0519  2 ! last offset in the list.
;  282          0520  2 !
;  283          0521  2
;  284          0522  2 IF .opcode EQL op_caseb                     ! If CASE instruction,
;  285          0523  2     OR .opcode EQL op_casew
;  286          0524  2     OR .opcode EQL op_casel
```

```
;  287          0525  2 THEN
;  288          0526  2     stream_ptr = .stream_ptr + (.last_literal_value+1)*2;
;  289          0527  2
;  290          0528  2 !
;  291          0529  2 !  Return a pointer to the beginning of the next instruction.
;  292          0530  2 !
;  293          0531  2
;  294          0532  2 IF NOT NULLPARAMETER(3)                 ! If RETLEN specified,
;  295          0533  2 THEN
;  296          0534  2     retlen [0] = .user_buffer_size - .user_buffer_left;
;  297          0535  2
;  298          0536  2 stream_pntr [0] = .stream_ptr;          ! Return pointer to next instruction
;  299          0537  2
;  300          0538  2 RETURN ss$_normal;
;  301          0539  2
;  302          0540  1 END;


                                                        .TITLE  LIB$INS_DECODE Instruction decoder
                                                        .IDENT  \V04-000\

                                                        .PSECT  Z$DEBUG_CODE,  PIC,2

                                                00000 USER_SYMBOLIZE_ROUTINE:
                                                        .BLKB   4
                                                00004 USER_BUFFER_ADDRESS:
                                                        .BLKB   4
                                                00008 USER_BUFFER_SIZE:
                                                        .BLKB   2
                                                0000A USER_BUFFER_LEFT:
                                                        .BLKB   2
                                                0000C LAST_LITERAL_VALUE:
                                                        .BLKB   4
23 20 4B 43 45 48 43 5F 47 55 42  00010 P.AAA:  .ASCII  \BUG_CHECK #\                          ;
                          20 20   0001B P.AAB:  .ASCII  \  \                                  ;
                             2C   0001D P.AAC:  .ASCII  \,\                                   ;

                                                        .EXTRN  LIB$GB_OPINFO, LIB$GB_OPINFO1
                                                        .EXTRN  LIB$GB_OPINFO2

                                          0FFC 00000            .ENTRY  LIB$INS_DECODE, Save R2,R3,R4,R5,R6,R7,R8,-  ; 0411
                                                                        R9,R10,R11
                                59      0000V  CF   9E 00002            MOVAB   APPEND_STRING, R9                    ;
                                58 00000000G   EF   9E 00007            MOVAB   LIB$GB_OPINFO2, R8                   ;
                                57 00000000G   EF   9E 0000E            MOVAB   LIB$GB_OPINFO1, R7                   ;
                                56        D2   AF   9E 00015            MOVAB   USER_BUFFER_SIZE, R6                 ;
                                5B        04   BC   D0 00019            MOVL    @STREAM_PNTR, STREAM_PTR             ; 0448
                                50        08   AC   D0 0001D            MOVL    OUTBUF, R0                           ; 0450
                                66             60   B0 00021            MOVW    (R0), USER_BUFFER_SIZE               ;
                             FC    A6     04   A0   B0 00024            MOVL    4(R0), USER_BUFFER_ADDRESS           ; 0451
                             02    A6          66   B0 00029            MOVW    USER_BUFFER_SIZE, USER_BUFFER_LEFT   ; 0452
                                04             6C   91 0002D            CMPB    (AP), #4                             ; 0454
                                               05   1F 00030            BLSSU   1$                                   ;
                                         10    AC   D5 00032            TSTL    16(AP)                               ;
                                               05   12 00035            BNEQ    2$                                   ;
                                         F8    A6   D4 00037 1$:        CLRL    USER_SYMBOLIZE_ROUTINE               ; 0456
                                               05   11 0003A            BRB     3$                                   ;
```

```
              F8   A6        10   AC  D0 0003C 2$:   MOVL    SYMBOLIZE_RTN, USER_SYMBOLIZE_ROUTINE       ; 0458
                   54             6B  9B 00041 3$:   MOVZBW  (STREAM_PTR), OPCODE                        ; 0466
            00FD   8F             54  B1 00044       CMPW    OPCODE, #253                                ; 0468
                                  0D  12 00049       BNEQ    4$                                          ;
                   50        01   AB  9A 0004B       MOVZBL  1(STREAM_PTR), R0                           ; 0471
        50         50             08  78 0004F       ASHL    #8, R0, R0                                  ;
                   54             50  A0 00053       ADDW2   R0, OPCODE                                  ;
                                  5B  D6 00056       INCL    STREAM_PTR                                  ; 0472
                   52             54  3C 00058 4$:   MOVZWL  OPCODE, R2                                  ; 0475
            00FF   8F             52  B1 0005B       CMPW    R2, #255                                    ;
                                  33  12 00060       BNEQ    6$                                          ;
              FE   8F        01   AB  91 00062       CMPB    1(STREAM_PTR), #254                         ; 0476
                                  2C  12 00067       BNEQ    6$                                          ;
                   51        08   A6  9E 00069       MOVAB   P.AAA, R1                                   ; 0480
                   50             0B  D0 0006D       MOVL    #11, R0                                     ;
                                  69  16 00070       JSB     APPEND_STRING                               ;
                                  02  DD 00072       PUSHL   #2                                          ; 0481
                   7E        02   AB  3C 00074       MOVZWL  2(STREAM_PTR), -(SP)                        ;
           0000V   CF             02  FB 00078       CALLS   #2, APPEND_HEX                              ;
              04   BC        04   AB  9E 0007D       MOVAB   4(R11), @STREAM_PNTR                        ; 0482
                   03             6C  91 00082       CMPB    (AP), #3                                    ; 0483
                                  0B  1F 00085       BLSSU   5$                                          ;
                             0C   AC  D5 00087       TSTL    12(AP)                                      ;
                                  06  13 0008A       BEQL    5$                                          ;
    0C  BC         66        02   A6  A3 0008C       SUBW3   USER_BUFFER_LEFT, USER_BUFFER_SIZE, @RETLEN ; 0485
                                00FA  31 00092 5$:   BRW     21$                                         ; 0486
                   51             04  D0 00095 6$:   MOVL    #4, OFFSET                                  ; 0489
                                  53  D4 00098       CLRL    R3                                          ;
              FD   8F             52  91 0009A       CMPB    R2, #253                                    ;
                                  0B  13 0009E       BEQL    7$                                          ;
                                  53  D6 000A0       INCL    R3                                          ;
                   50           6142  7E 000A2       MOVAQ   (OFFSET)[R2], R0                            ;
                   50             57  C0 000A6       ADDL2   R7, R0                                      ;
                                  0C  11 000A9       BRB     8$                                          ;
        50         52        F8   8F  78 000AB 7$:   ASHL    #-8, R2, R0                                 ;
                   50           6140  7E 000B0       MOVAQ   (OFFSET)[R0], R0                            ;
                   50             58  C0 000B4       ADDL2   R8, R0                                      ;
  0F    60         04             00  ED 000B7 8$:   CMPZV   #0, #4, (R0), #15                           ;
                                  04  12 000BC       BNEQ    9$                                          ;
                   50             02  D0 000BE       MOVL    #2, R0                                      ; 0491
                                      04 000C1       RET                                                 ;
                                  5B  D6 000C2 9$:   INCL    STREAM_PTR                                  ; 0498
                                  51  D4 000C4       CLRL    OFFSET                                      ; 0500
                   09             53  E9 000C6       BLBC    R3, 10$                                     ;
                   50           6142  7E 000C9       MOVAQ   (OFFSET)[R2], R0                            ;
                   50             57  C0 000CD       ADDL2   R7, R0                                      ;
                                  0C  11 000D0       BRB     11$                                         ;
        50         52        F8   8F  78 000D2 10$:  ASHL    #-8, R2, R0                                 ;
                   50           6140  7E 000D7       MOVAQ   (OFFSET)[R0], R0                            ;
                   50             58  C0 000DB       ADDL2   R8, R0                                      ;
                                  50  DD 000DE 11$:  PUSHL   R0                                          ;
                                  02  DD 000E0       PUSHL   #2                                          ;
           0000V   CF             02  FB 000E2       CALLS   #2, APPEND_RAD50                            ;
                   51        13   A6  9E 000E7       MOVAB   P.AAB, R1                                   ; 0501
                   50             02  D0 000EB       MOVL    #2, R0                                      ;
                                  69  16 000EE       JSB     APPEND_STRING                               ;
                   51             04  D0 000F0       MOVL    #4, OFFSET                                  ; 0507
```

```
                                52              54  3C 000F3          MOVZWL  OPCODE, R2                                      ;
                                                55  D4 000F6          CLRL    R5                                              ;
                             FD 8F                52  91 000F8          CMPB    R2, #253                                        ;
                                                  0B  13 000FC          BEQL    12$                                             ;
                                                55  D6 000FE          INCL    R5                                              ;
                                50            6142  7E 00100          MOVAQ   (OFFSET)[R2], R0                                ;
                                50              57  C0 00104          ADDL2   R7, R0                                          ;
                                                0C  11 00107          BRB     13$                                             ;
                    50          52        F8    8F  78 00109 12$:     ASHL    #-8, R2, R0                                     ;
                                50            6140  7E 0010E          MOVAQ   (OFFSET)[R0], R0                                ;
                                50              58  C0 00112          ADDL2   R8, R0                                          ;
              54    60          04              00  EF 00115 13$:     EXTZV   #0, #4, (R0), R4                                ;
                                                53  D4 0011A          CLRL    I                                               ;
                                                3B  11 0011C          BRB     17$                                             ;
                                                52  DD 0011E 14$:     PUSHL   R2                                              ; 0510
                                                53  DD 00120          PUSHL   I                                               ;
                          0000V CF                02  FB 00122          CALLS   #2, INS_OPERAND                                 ;
                                68                50  E9 00127          BLBC    STATUS, 22$                                     ;
                                                53  D5 0012A          TSTL    I                                               ; 0512
                                                2B  13 0012C          BEQL    17$                                             ;
                                51              04  D0 0012E          MOVL    #4, OFFSET                                      ;
                                09              55  E9 00131          BLBC    R5, 15$                                         ;
                                50            6142  7E 00134          MOVAQ   (OFFSET)[R2], R0                                ;
                                50              57  C0 00138          ADDL2   R7, R0                                          ;
                                                0C  11 0013B          BRB     16$                                             ;
                    50          52        F8    8F  78 0013D 15$:     ASHL    #-8, R2, R0                                     ;
                                50            6140  7E 00142          MOVAQ   (OFFSET)[R0], R0                                ;
                                50              58  C0 00146          ADDL2   R8, R0                                          ;
              53    60          04              00  ED 00149 16$:     CMPZV   #0, #4, (R0), I                                 ;
                                                09  15 0014E          BLEQ    17$                                             ;
                                51        15    A6  9E 00150          MOVAB   P.AAC, R1                                       ; 0514
                                50              01  D0 00154          MOVL    #1, R0                                          ;
                                                69  16 00157          JSB     APPEND_STRING                                   ;
                    C1          53              54  F3 00159 17$:     AOBLEQ  R4, I, 14$                                      ; 0507
                          008F  8F              52  B1 0015D          CMPW    R2, #143                                        ; 0522
                                                0E  13 00162          BEQL    18$                                             ;
                          00AF  8F              52  B1 00164          CMPW    R2, #175                                        ; 0523
                                                07  13 00169          BEQL    18$                                             ;
                          00CF  8F              52  B1 0016B          CMPW    R2, #207                                        ; 0524
                                                09  12 00170          BNEQ    19$                                             ;
                                50        04    A6  D0 00172 18$:     MOVL    LAST_LITERAL_VALUE, R0                          ; 0526
                                5B        02 AB40   3E 00176          MOVAW   2(STREAM_PTR)[R0], STREAM_PTR                   ;
                                03              6C  91 0017B 19$:     CMPB    (AP), #3                                        ; 0532
                                                0B  1F 0017E          BLSSU   20$                                             ;
                                              0C    AC  D5 00180          TSTL    12(AP)                                          ;
                                                06  13 00183          BEQL    20$                                             ;
                 0C   BC                      66    02  A6  A3 00185          SUBW3   USER_BUFFER_LEFT, USER_BUFFER_SIZE, @RETLEN     ; 0534
                          04    BC                    5B  D0 0018B 20$:     MOVL    STREAM_PTR, @STREAM_PNTR                        ; 0536
                                50                    01  D0 0018F 21$:     MOVL    #1, R0                                          ; 0538
                                                        04 00192 22$:     RET                                                     ; 0540

; Routine Size:  403 bytes,    Routine Base:  Z$DEBUG_CODE + 001E
```

```
;  304          0541  1 %SBTTL 'INS_OPERAND - Output instruction''s operand'
;  305          0542  1 ROUTINE ins_operand(index, opcode): ptr_linkage =
;  306          0543  1
;  307          0544  1 !---
;  308          0545  1 !       Print out a reference to an instruction operand.
;  309          0546  1 !
;  310          0547  1 ! Warning:
;  311          0548  1 !
;  312          0549  1 !       1) there is code in the 'deferred' macro which will cease
;  313          0550  1 !          to work when/if we change the representation of true
;  314          0551  1 !          and false.
;  315          0552  1 !       2) the local macros, below, check for the indicated addressing
;  316          0553  1 !          modes only given that they appear in the code where they
;  317          0554  1 !          do - ie, the checks take advantage of what we know about
;  318          0555  1 !          which cases we have already eliminated, etc.
;  319          0556  1 !
;  320          0557  1 ! Inputs:
;  321          0558  1 !
;  322          0559  1 !       stream_ptr = a byte pointer to the first byte of the instruction
;  323          0560  1 !                    stream which begins the reference to this operand.
;  324          0561  1 !                    this byte is what we refer to as the dominant mode.
;  325          0562  1 !       index = ordinal of which operand we are on.  this is needed to
;  326          0563  1 !                    decide the 'context' for this operand if pc-relative
;  327          0564  1 !                    addressing mode is used.
;  328          0565  1 !       opcode = The opcode we are currently working on.
;  329          0566  1 !                (This parameter has already been validated.)
;  330          0567  1 !
;  331          0568  1 ! Outputs:
;  332          0569  1 !
;  333          0570  1 !       R0 = status code
;  334          0571  1 !       The stream_ptr is incremented to reflect how much of the instruction
;  335          0572  1 !       stream we have 'eaten up'.  This pointer should point to the beginning
;  336          0573  1 !       of either the next instruction, or the next operand reference,
;  337          0574  1 !       depending on how many operands the current instruction has.
;  338          0575  1 !---
;  339          0576  1
;  340          0577  2 BEGIN
;  341          0578  2
;  342          0579  2 !
;  343          0580  2 ! Local macros used to check for the indicated addressing modes.
;  344          0581  2 !
;  345          0582  2
;  346          0583  2 MACRO
;  347          0584  2     registr(mode)                   ! register mode addressing.
;  348          0585  2         = (mode EQL 5) %,
;  349          0586  2     deferred(mode)                  ! those which begin with '@' are
;  350          0587  2         = (mode LSS 0 AND mode)%,
;  351          0588  2                                     !  9 - @(rn)+,
;  352          0589  2                                     !  b - @byte(rn),
;  353          0590  2                                     !  d - @word(rn),
;  354          0591  2                                     !  f - @long(rn),
;  355          0592  2                                     ! or any of these + indexing.
;  356          0593  2                                     ! the thing which is common to only these
;  357          0594  2                                     ! modes is that they all have the sign
;  358          0595  2                                     ! bit set and are odd!
;  359          0596  2
;  360          0597  2     autodec(mode)                   ! see if mode is auto decrement.
```

```
;  361          0598  2          = (mode EQL 7)%,          ! this check is right from srm.
;  362          0599  2
;  363          0600  2      autoinc(mode)                 ! mode is auto increment
;  364          0601  2          = (mode LSS 0)%;
;  365          0602  2                                    ! this check depends upon the fact that
;  366          0603  2                                    ! we extracted the mode with sign extension,
;  367          0604  2                                    ! and that we have already eliminated
;  368          0605  2                                    ! many of the other possibilities.
;  369          0606  2
;  370          0607  2  EXTERNAL REGISTER
;  371          0608  2      stream_ptr=11: REF BLOCK [,BYTE];   ! Points to the instruction stream
;  372          0609  2
;  373          0610  2  LOCAL
;  374          0611  2      flag,                         ! indicates which type of displacment .  have.
;  375          0612  2      displ,                        ! the actual displacement.
;  376          0613  2      disp_size,                    ! the size, in bytes, of a displacement.
;  377          0614  2      dom_oprnd,                    ! operand which we extract from the
;  378          0615  2                                    !  dominant mode byte.  it may be rn,
;  379          0616  2                                    !  rx, or a literal.  (srm notation).
;  380          0617  2      dom_mode;                     ! the primary addressing mode comes from
;  381          0618  2                                    !    this dominant byte as well.
;  382          0619  2
;  383          0620  2  !
;  384          0621  2  ! We have to consider the possibility of
;  385          0622  2  ! so-called 'branch type' addressing first
;  386          0623  2  ! before anything else because otherwise you cannot
;  387          0624  2  ! differentiate short literal from byte displacement
;  388          0625  2  ! branching.
;  389          0626  2  !
;  390          0627  2
;  391          0628  2  IF branch_type(.index, .opcode)          ! If we can output branch operand,
;  392          0629  2  THEN
;  393          0630  2      RETURN ss$_normal;                  ! Return with updated stream pointer
;  394          0631  2
;  395          0632  2  !
;  396          0633  2  ! See that we can access at least the operand byte.
;  397          0634  2  !
;  398          0635  2
;  399          0636  2  probe(.stream_ptr, 1);                 ! Return if we can't read the operand
;  400          0637  2
;  401          0638  2  !
;  402          0639  2  ! Extract the needed fields from the first byte of
;  403          0640  2  !  the operand specifier.  We extract some fields
;  404          0641  2  !  with sign extension simply because that makes
;  405          0642  2  !  making various tests more convenient.
;  406          0643  2  !
;  407          0644  2
;  408          0645  2  dom_mode = .stream_ptr [amode];
;  409          0646  2  dom_oprnd = .stream_ptr [areg];
;  410          0647  2
;  411          0648  2  !
;  412          0649  2  ! Take special action for indexing mode.
;  413          0650  2  !
;  414          0651  2
;  415          0652  2  IF .dom_mode EQL indexing_mode
;  416          0653  2  THEN
;  417          0654  3      BEGIN
```

```
;  418          0655  3      ! handle indexing mode recursively.
;  419          0656  3
;  420          0657  3      stream_ptr = stream_ptr [next_field(1)];
;  421          0658  3      return_if_error(ins_operand(.index, .opcode));
;  422          0659  3      put_reg(.dom_oprnd, square_brackets);
;  423          0660  3      RETURN ss$_normal;
;  424          0661  2      END;
;  425          0662  2
;  426          0663  2 ! Simple modes are easier:
;  427          0664  2
;  428          0665  2 ! First see if there will be a literal or displacement in the operand.
;  429          0666  2
;  430          0667  2 return_if_error(displacement(flag, displ, disp_size, .index, .opcode));
;  431          0668  2
;  432          0669  2 ! Begin checking for the addressing modes which begin
;  433          0670  2 ! with special characters since we have to print them
;  434          0671  2 ! first.  We try to handle similar cases with the same
;  435          0672  2 ! code, getting the differences out of the way first.
;  436          0673  2
;  437          0674  3 IF deferred(.dom_mode)
;  438          0675  2 THEN
;  439          0676  2     append('@')
;  440          0677  2 ELSE
;  441          0678  3     IF autodec(.dom_mode)
;  442          0679  2     THEN
;  443          0680  2         append('-');
;  444          0681  2
;  445          0682  2 ! Next we have to consider displacements or literals.
;  446          0683  2 ! Whether or not this is the case has already been
;  447          0684  2 ! determined in the call to 'displacement', above.
;  448          0685  2
;  449          0686  2 IF .flag
;  450          0687  2 THEN
;  451          0688  3     BEGIN
;  452          0689  3     ! There is a literal, so print it.
;  453          0690  3     ! The flag value returned by displacement()
;  454          0691  3     ! distinguishes when there should be a '#',
;  455          0692  3     ! as opposed to when the number is actually
;  456          0693  3     ! a displacement off a register.
;  457          0694  3
;  458          0695  3     IF .flag GTR 0                          ! If its a literal,
;  459          0696  3     THEN
;  460          0697  4         BEGIN
;  461          0698  4         append('#');
;  462          0699  4
;  463          0700  4         ! except for @# mode, Make .dom_oprnd neq pc_reg so that
;  464          0701  4         ! later only checking that will also tell us
;  465          0702  4         ! that .flag is gtr 0.
;  466          0703  4
;  467          0704  5         IF not deferred(.dom_mode)
;  468          0705  4         THEN
;  469          0706  4             dom_oprnd = pc_reg +1;
;  470          0707  4 !       END
;  471          0708  4 !    ELSE                                   ! Else, for displacements,
;  472          0709  4 !       BEGIN
;  473          0710  4 !       OWN
;  474          0711  4 !           displ_id: VECTOR[4,BYTE]
```

```
;  475          0712  4 !                          INITIAL( BYTE( 'B', 'W', '?', 'L') );
;  476          0713  4 !
;  477          0714  4 !       ! Print an indication of the displacement size.
;  478          0715  4 !
;  479          0716  4 !       append_string(1, displ_id [.disp_size-1]);
;  480          0717  4 !       append('^');
;  481          0718  3         END;
;  482          0719  3
;  483          0720  3     ! Output here is always "displ(reg)", for non-PC
;  484          0721  3     ! displacments, and just "effective", otherwise.
;  485          0722  3
;  486          0723  3     IF .dom_oprnd EQL pc_reg                    ! If PC relative or absolute,
;  487          0724  3     THEN
;  488          0725  4         BEGIN
;  489          0726  4         IF .flag LSS 0                          ! If PC relative,
;  490          0727  4         THEN
;  491          0728  5             BEGIN
;  492          0729  5             disp_size = 4;
;  493          0730  5             displ = .displ + .stream_ptr;       ! Make an effective address
;  494          0731  5             append_address(.displ, 0);          ! Output relative address
;  495          0732  5             END
;  496          0733  4         ELSE                                    ! Else, if absolute address,
;  497          0734  4             append_address(.displ, 1);          ! Output absolute address
;  498          0735  4         END
;  499          0736  3     ELSE
;  500          0737  4         BEGIN
;  501          0738  4
;  502          0739  4         ! Literals or real (non-PC) displacement modes.
;  503          0740  4
;  504          0741  4         append_hex(.displ, .disp_size);         ! Output literal or offset,
;  505          0742  4         last_literal_value = .displ;            ! Save last literal value
;  506          0743  4
;  507          0744  4         IF .flag LSS 0                          ! If relative (from register),
;  508          0745  4         THEN
;  509          0746  4             put_reg(.dom_oprnd, round_brackets);
;  510          0747  3         END;
;  511          0748  3     END
;  512          0749  3
;  513          0750  3 ! No literal or displacement -> we must have some type of
;  514          0751  3 ! register reference.  Sort out the few cases and print them.
;  515          0752  3
;  516          0753  2 ELSE
;  517          0754  3     IF registr(.dom_mode)
;  518          0755  2     THEN
;  519          0756  2         put_reg(.dom_oprnd, no_brackets)
;  520          0757  2     ELSE
;  521          0758  3         BEGIN
;  522          0759  3         put_reg(.dom_oprnd, round_brackets);
;  523          0760  4         IF autoinc(.dom_mode)
;  524          0761  3         THEN
;  525          0762  3             append('+');
;  526          0763  2         END;
;  527          0764  2
;  528          0765  2 RETURN ss$_normal;
;  529          0766  2
;  530          0767  1 END;
```

```
                                                40  001B1 P.AAD:   .ASCII  \@\                                                    ;
                                                2D  001B2 P.AAE:   .ASCII  \-\                                                    ;
                                                23  001B3 P.AAF:   .ASCII  \#\                                                    ;
                                                2B  001B4 P.AAG:   .ASCII  \+\                                                    ;


                                                    07FC 00000 INS_OPERAND:
                                                               .WORD   Save R2,R3,R4,R5,R6,R7,R8,R9,R10                           ; 0542
                                    56      0000V   CF   9E 00002      MOVAB   APPEND_STRING, R6                                  ;
                                    55        F6    AF   9E 00007      MOVAB   INS_OPERAND, R5                                    ;
                                    5E              0C   C2 0000B      SUBL2   #12, SP                                            ;
                                    7E        04    AC   7D 0000E      MOVQ    INDEX, -(SP)                                       ; 0628
                              0000V CF              02   FB 00012      CALLS   #2, BRANCH_TYPE                                    ;
                                    03              50   E9 00017      BLBC    R0, 1$                                             ;
                                                00D0  31   0001A      BRW     16$                                                ;
                  52        68                  04   EE 0001D 1$:    EXTV    #4, #4, (STREAM_PTR), DOM_MODE                     ; 0645
                  54        68                  04   EF 00022      EXTZV   #0, #4, (STREAM_PTR), DOM_OPRND                    ; 0646
                                    04              52   D1 00027      CMPL    DOM_MODE, #4                                       ; 0652
                                                11   12 0002A      BNEQ    2$                                                 ;
                                                5B   D6 0002C      INCL    STREAM_PTR                                         ; 0657
                                    7E        04    AC   7D 0002E      MOVQ    INDEX, -(SP)                                       ; 0658
                                    65              02   FB 00032      CALLS   #2, INS_OPERAND                                    ;
                                    17              50   E9 00035      BLBC    STATUS, 3$                                         ;
                                                    02   DD 00038      PUSHL   #2                                                 ; 0659
                                                0091  31   0003A      BRW     14$                                                ;
                                    7E        04    AC   7D 0003D 2$:    MOVQ    INDEX, -(SP)                                       ; 0667
                                              08    AE   9F 00041      PUSHAB  DISP_SIZE                                          ;
                                              10    AE   9F 00044      PUSHAB  DISP[                                              ;
                                              18    AE   9F 00047      PUSHAB  FLAG                                               ;
                              0000V CF              05   FB 0004A      CALLS   #5, DISPLACEMENT                                   ;
                                    01              50   E8 0004F 3$:    BLBS    STATUS, 4$                                         ;
                                                    04   00052        RET                                                        ;
                                                53   D4 00053 4$:    CLRL    R3                                                 ; 0674
                                                52   D5 00055      TSTL    DOM_MODE                                           ;
                                                0B   18 00057      BGEQ    5$                                                 ;
                                                53   D6 00059      INCL    R3                                                 ;
                                    06              52   E9 0005B      BLBC    DOM_MODE, 5$                                       ;
                                    51        9B    AF   9E 0005E      MOVAB   P.AAD, R1                                          ; 0676
                                                09   11 00062      BRB     6$                                                 ;
                                    07              52   D1 00064 5$:    CMPL    DOM_MODE, #7                                       ; 0678
                                                09   12 00067      BNEQ    7$                                                 ;
                                    51        91    AF   9E 00069      MOVAB   P.AAE, R1                                          ; 0680
                                    50              01   D0 0006D 6$:    MOVL    #1, R0                                             ;
                                                66   16 00070      JSB     APPEND_STRING                                      ;
                                    51        08    AE   E9 00072 7$:    BLBC    FLAG, T3$                                          ; 0686
                                              08    AE   D5 00076      TSTL    FLAG                                               ; 0695
                                                12   15 00079      BLEQ    9$                                                 ;
                                    51        80    AF   9E 0007B      MOVAB   P.AAF, R1                                          ; 0698
                                    50              01   D0 0007F      MOVL    #1, R0                                             ;
                                                66   16 00082      JSB     APPEND_STRING                                      ;
                                    03              53   E9 00084      BLBC    R3, 8$                                             ; 0704
                                    03              52   E8 00087      BLBS    DOM_MODE, 9$                                       ;
                                    54              10   D0 0008A 8$:    MOVL    #16, DOM_OPRND                                     ; 0706
                                    0F              54   D1 0008D 9$:    CMPL    DOM_OPRND, #15                                     ; 0723
```

```
                              1C  12 00090         BNEQ    12$                                   ;
                      08      AE  D5 00092         TSTL    FLAG                                  ; 0726
                              0B  18 00095         BGEQ    10$                                   ;
               6E             04  D0 00097         MOVL    #4, DISP_SIZE                         ; 0729
       04      AE             5B  C0 0009A         ADDL2   STREAM_PTR, DISPL                     ; 0730
                              7E  D4 0009E         CLRL    -(SP)                                 ; 0731
                              02  11 000A0         BRB     11$                                   ;
                              01  DD 000A2 10$:    PUSHL   #1                                    ; 0734
                      08      AE  DD 000A4 11$:    PUSHL   DISPL                                 ;
    0000V      CF             02  FB 000A7         CALLS   #2, APPEND_ADDRESS                    ;
                              3F  11 000AC         BRB     16$                                   ; 0723
                              6E  DD 000AE 12$:    PUSHL   DISP_SIZE                             ; 0741
                      08      AE  DD 000B0         PUSHL   DISPL                                 ;
    0000V      CF             02  FB 000B3         CALLS   #2, APPEND_HEX                        ;
    FD99       CF     04      AE  D0 000B8         MOVL    DISPL, LAST_LITERAL_VALUE             ; 0742
                      08      AE  D5 000BE         TSTL    FLAG                                  ; 0744
                              2A  18 000C1         BGEQ    16$                                   ;
                              7E  D4 000C3         CLRL    -(SP)                                 ; 0746
                              07  11 000C5         BRB     14$                                   ;
               05             52  D1 000C7 13$:    CMPL    DOM_MODE, #5                          ; 0754
                              0B  12 000CA         BNEQ    15$                                   ;
                              01  DD 000CC         PUSHL   #1                                    ; 0756
                              54  DD 000CE 14$:    PUSHL   DOM_OPRND                             ;
    0000V      CF             02  FB 000D0         CALLS   #2, PUT_REG                           ;
                              16  11 000D5         BRB     16$                                   ;
                              7E  D4 000D7 15$:    CLRL    -(SP)                                 ; 0759
                              54  DD 000D9         PUSHL   DOM_OPRND                             ;
    0000V      CF             02  FB 000DB         CALLS   #2, PUT_REG                           ;
               0A             53  E9 000E0         BLBC    R3, 16$                               ; 0760
               51     FF18    CF  9E 000E3         MOVAB   P.AAG, R1                             ; 0762
               50             01  D0 000E8         MOVL    #1, R0                                ;
                              66  16 000EB         JSB     APPEND_STRING                         ;
               50             01  D0 000ED 16$:    MOVL    #1, R0                                ; 0765
                                  04 000F0         RET                                           ; 0767

; Routine Size:  241 bytes,    Routine Base:  Z$DEBUG_CODE + 01B5
```

```
;  532          0768  1 %SBTTL 'BRANCH_TYPE - Handle branch operands'
;  533          0769  1 ROUTINE branch_type(index, opcode): ptr_linkage =
;  534          0770  1
;  535          0771  1 !---
;  536          0772  1 !       Decide if the current operand is using branch type
;  537          0773  1 !       addressing.  If so, print out the reference and
;  538          0774  1 !       look after all the details.  Otherwise return 0
;  539          0775  1 !       and let someone else handle it.
;  540          0776  1 !
;  541          0777  1 ! Inputs:
;  542          0778  1 !
;  543          0779  1 !       stream_ptr = a pointer to the current dominant mode byte.
;  544          0780  1 !       index = which operand (ordinal) we're working on.
;  545          0781  1 !       opcode = The opcode we are currently working on.
;  546          0782  1 !                (This parameter has already been validated.)
;  547          0783  1 !
;  548          0784  1 ! Routine value:
;  549          0785  1 !
;  550          0786  1 !       Routine value is true if the current operand is a branch, else false.
;  551          0787  1 !
;  552          0788  1 !       If the current operand is a branch, the reference is appended
;  553          0789  1 !       to the output buffer and the stream pointer is updated.
;  554          0790  1 !--
;  555          0791  1
;  556          0792  2 BEGIN
;  557          0793  2
;  558          0794  2 EXTERNAL REGISTER
;  559          0795  2     stream_ptr=11;                  ! Points to the instruction stream
;  560          0796  2
;  561          0797  2 LOCAL
;  562          0798  2     n_ops,                          ! number of operands for current opcode
;  563          0799  2     disp_size,                      ! size of branch operand, in bytes.
;  564          0800  2     displ;                          ! the actual branch displacement.
;  565          0801  2
;  566          0802  2 ! There is no point in even considering branch type
;  567          0803  2 ! addressing unless we're on the last operand for
;  568          0804  2 ! this instruction.
;  569          0805  2
;  570          0806  2 n_ops = .lib$gb_opinfo [.opcode, op_numops];
;  571          0807  2
;  572          0808  2 IF .n_ops NEQ .index
;  573          0809  2 THEN
;  574          0810  2     RETURN false;
;  575          0811  2
;  576          0812  2 ! Now we know we can take the op_br_type field literally.
;  577          0813  2 ! it contains the number of bytes used for the branch
;  578          0814  2 ! displacement.  0 in this field indicates that
;  579          0815  2 ! this opcode has no branch type operands.
;  580          0816  2
;  581          0817  2 disp_size = .lib$gb_opinfo [.opcode, op_br_type];
;  582          0818  2
;  583          0819  2 IF .disp_size EQL no_branch
;  584          0820  2 THEN
;  585          0821  2     RETURN false;
;  586          0822  2
;  587          0823  2 probe(.stream_ptr,.disp_size);          ! Exit if we can't read displacement
;  588          0824  2
```

```
;  589          0825  2 !
;  590          0826  2 ! Success!  We have discovered a case of branch type addressing.
;  591          0827  2 ! handle this by simply extracting the field, (with sign
;  592          0828  2 ! extension as per srm),  printing out the reference,
;  593          0829  2 ! and returning a pointer to the next instruction.
;  594          0830  2 !
;  595          0831  2
;  596          0832  2 displ = .(.stream_ptr)<0,.disp_size*8,1>;
;  597          0833  2 stream_ptr = .stream_ptr + .disp_size;
;  598          0834  2
;  599          0835  2 append_address(.stream_ptr + .displ, 0); ! Output relative address
;  600          0836  2
;  601          0837  2 RETURN true;
;  602          0838  2
;  603          0839  1 END;


                                        003C 00000 BRANCH_TYPE:
                                                   .WORD   Save R2,R3,R4,R5                        ; 0769
                              55 00000000G   EF  9E 00002  MOVAB   LIB$GB_OPINF01, R5
                              54 00000000G   EF  9E 00009  MOVAB   LIB$GB_OPINF02, R4
                              52             04  D0 00010  MOVL    #4, OFFSET                      ; 0806
                              51       08    AC  D0 00013  MOVL    OPCODE, R1
                                             53  D4 00017  CLRL    R3
                           FD 8F             51  91 00019  CMPB    R1, #253
                                             0B  13 0001D  BEQL    1$
                                             53  D6 0001F  INCL    R3
                              50           6241  7E 00021  MOVAQ   (OFFSET)[R1], R0
                              50             55  C0 00025  ADDL2   R5, R0
                                             0C  11 00028  BRB     2$
                        50                   51  F8 8F 78 0002A 1$: ASHL    #-8, R1, R0
                              50           6240  7E 0002F  MOVAQ   (OFFSET)[R0], R0
                              50             54  C0 00033  ADDL2   R4, R0
                     50       60             04  00 EF 00036 2$:  EXTZV   #0, #4, (R0), N_OPS
                                 04          AC  50 D1 0003B  CMPL    N_OPS, INDEX                    ; 0808
                                             3C  12 0003F  BNEQ    5$
                              52             07  D0 00041  MOVL    #7, OFFSET                      ; 0817
                              09             53  E9 00044  BLBC    R3, 3$
                              50           6241  7E 00047  MOVAQ   (OFFSET)[R1], R0
                              50             55  C0 0004B  ADDL2   R5, R0
                                             0C  11 0004E  BRB     4$
                        50                   51  F8 8F 78 00050 3$:  ASHL    #-8, R1, R0
                              50           6240  7E 00055  MOVAQ   (OFFSET)[R0], R0
                              50             54  C0 00059  ADDL2   R4, R0
                     50       60             04  04 EF 0005C 4$:  EXTZV   #4, #4, (R0), DISP_SIZE
                                             1A  13 00061  BEQL    5$                              ; 0819
                        51                   50  03 78 00063  ASHL    #3, DISP_SIZE, R1               ; 0832
                     52 6B                   51  00 EE 00067  EXTV    #0, R1, (STREAM_PTR), DISPL
                                             5B  50 C0 0006C  ADDL2   DISP_SIZE, STREAM_PTR           ; 0833
                                             7E  D4 0006F  CLRL    -(SP)                           ; 0835
                                           624B  9F 00071  PUSHAB  (DISPL)[STREAM_PTR]
                                 0000V       CF  02 FB 00074  CALLS   #2, APPEND_ADDRESS
                                             50  01 D0 00079  MOVL    #1, R0                          ; 0837
                                                 04 0007C  RET
```

```
                                                50  D4 0007D 5$:    CLRL    R0                              ; 0839
                                                    04 0007F        RET                                     ;

; Routine Size:  128 bytes,    Routine Base:  Z$DEBUG_CODE + 02A6
```

```
;  605          0840  1 %SBTTL 'DISPLACEMENT - Determine size of operand'
;  606          0841  1 ROUTINE displacement (flag, displ, ptr_disp_size, index, opcode): ptr_linkage =
;  607          0842  1
;  608          0843  1 !---
;  609          0844  1 !       Return any displacement associated with the current operand of the
;  610          0845  1 !       current instruction. Note that for short literals, the literal is returned
;  611          0846  1 !       in DISPL; for displacement mode instructions, the actual displacement is
;  612          0847  1 !       returned in DISPL; and for PC Mode instructions, the displacement is returned
;  613          0848  1 !       in DISPL. For other mode instructions, the routine effectively No-ops.
;  614          0849  1 !
;  615          0850  1 ! Inputs:
;  616          0851  1 !
;  617          0852  1 !       stream_ptr = Where the current operand specifier starts.
;  618          0853  1 !       flag = Where we indicate the displacement type
;  619          0854  1 !       displ = Where we put the actual displacement
;  620          0855  1 !       ptr_disp_size = Number of bytes in the displcaement
;  621          0856  1 !       index = Designates the Current operand
;  622          0857  1 !       opcode = Number of opcode of current instruction
;  623          0858  1 !
;  624          0859  1 !
;  625          0860  1 ! Outputs:
;  626          0861  1 !
;  627          0862  1 !       R0 = status code
;  628          0863  1 !       flag = 1 if literal, -1 if displacement, 0 otherwise
;  629          0864  1 !       displ = Displacement or literal value
;  630          0865  1 !       ptr_disp_size = Number of bytes of displacement
;  631          0866  1 !
;  632          0867  1 !       The stream pointer is updated to the next operand or address
;  633          0868  1 !       of the same operand if a displacement wasn't found.
;  634          0869  1 !--
;  635          0870  1
;  636          0871  2 BEGIN
;  637          0872  2
;  638          0873  2 MACRO
;  639          0874  2     short_literal_mode = (.mode LEQ 3 AND .mode GEQ 0)%,
;  640          0875  2     displacement_mode = (.mode LEQ 15 AND .mode GEQ 10)%,
;  641          0876  2     pc_mode = (.reg EQL pc_reg AND (.mode EQL 8 OR .mode EQL 9))%;
;  642          0877  2
;  643          0878  2 EXTERNAL REGISTER
;  644          0879  2     stream_ptr=11: REF BLOCK [,BYTE];   ! Points to the instruction stream
;  645          0880  2
;  646          0881  2 MAP
;  647          0882  2     flag:        REF VECTOR,
;  648          0883  2     displ:       REF BLOCK,
;  649          0884  2     opcode:      BLOCK,
;  650          0885  2     ptr_disp_size: REF VECTOR;
;  651          0886  2
;  652          0887  2 LOCAL
;  653          0888  2     reg,                                ! Register in operand specifier
;  654          0889  2     mode;                               ! Mode in operand specifier
;  655          0890  2
;  656          0891  2 reg = .stream_ptr [areg];
;  657          0892  2 mode = .stream_ptr [dspl_mode];
;  658          0893  2
;  659          0894  2 ! Get past operand specifier byte
;  660          0895  2
;  661          0896  2 stream_ptr = stream_ptr [next_field(1)];
```

```
;  662          0897  2
;  663          0898  2 SELECTONE true OF
;  664          0899  2     SET
;  665          0900  2     [short_literal_mode]:                ! Short literal mode
;  666          0901  3                 BEGIN
;  667          0902  3                 ! Short literals only allowed on read-only operands
;  668          0903  3                 IF .lib$gb_opinfo [.opcode, op_datatype(.index)] NEQ access_r
;  669          0904  3                 THEN
;  670          0905  3                     RETURN lib$_noinstran; ! then return invalid instruction
;  671          0906  3                 ! Extract the number from operand specifier
;  672          0907  3                 displ [0,0,32,0] = .mode<0,2,0>^4 OR .reg;
;  673          0908  3                 flag [0] = 1;            ! Say its a literal
;  674          0909  3                 ptr_disp_size [0] = 1;
;  675          0910  3                 RETURN ss$_normal;
;  676          0911  2                 END;
;  677          0912  2     [displacement_mode]:                 ! Displacement modes
;  678          0913  3                 BEGIN
;  679          0914  3                 flag [0] = -1;           ! Say its a displacement
;  680          0915  3                 ptr_disp_size [0] =
;  681          0916  4                         (CASE .mode FROM 10 TO 15 OF
;  682          0917  4                             SET
;  683          0918  4                             [12,13]: 2;      ! 2 bytes of displacement info
;  684          0919  4                             [14,15]: 4;      ! 4 bytes of displacement info
;  685          0920  4                             [INRANGE]: 1;    ! 1 byte of displacement info
;  686          0921  3                             TES);
;  687          0922  3                 ! Save off the displacement
;  688          0923  3                 block [.displ,0,0,32,0] = .stream_ptr [0,0,8*.ptr_disp_size [0],1];
;  689          0924  3                 stream_ptr = stream_ptr [next_field(.ptr_disp_size [0])];
;  690          0925  3                 RETURN ss$_normal;
;  691          0926  2                 END;
;  692          0927  2     [pc_mode]:                           ! PC Modes
;  693          0928  3                 BEGIN
;  694          0929  3                 flag [0] = 1;            ! Say its a literal
;  695          0930  3                 IF .mode EQL 9
;  696          0931  3                 THEN
;  697          0932  3                     ptr_disp_size [0] = 4        ! 4 bytes of address
;  698          0933  3                 ! Else amount of displacement is dependent upon instruction
;  699          0934  3                 ELSE
;  700          0935  3                     ptr_disp_size [0] = ins_context(.index, .opcode);
;  701          0936  3                 block [.displ,0,0,32,0] = .stream_ptr [0,0,8*MIN(.ptr_disp_size [0], 4), 0];
;  702          0937  3                 IF .ptr_disp_size [0] GTR 4
;  703          0938  3                 THEN
;  704          0939  3                     RETURN lib$_numtrunc; ! Can't handle quad or octawords yet.
;  705          0940  3                 stream_ptr = stream_ptr [next_field(.ptr_disp_size [0])];
;  706          0941  3                 RETURN ss$_normal;
;  707          0942  2                 END;
;  708          0943  2     [OTHERWISE]:
;  709          0944  3                 BEGIN                    ! None of the above, so No op.
;  710          0945  3                 flag [0] = 0;            ! Not a displacement
;  711          0946  3                 ptr_disp_size [0] = 0;
;  712          0947  3                 displ [0,0,32,0] = 0;
;  713          0948  3                 ! Back over the byte we advanced over earlier
;  714          0949  3                 stream_ptr = stream_ptr [next_field(0)];
;  715          0950  3                 RETURN ss$_normal;
;  716          0951  2                 END;
;  717          0952  2     TES;
;  718          0953  2
```

```
;  719          0954  1 END;


                                        003C 00000 DISPLACEMENT:
                                                   .WORD   Save R2,R3,R4,R5                        ; 0841
        54        6B          04            00  EF 00002         EXTZV   #0, #4, (STREAM_PTR), REG       ; 0891
        50        8B          04            04  EF 00007         EXTZV   #4, #4, (STREAM_PTR)+, MODE     ; 0892
                                            52  D4 0000C         CLRL    R2                              ; 0900
                              03            50  D1 0000E         CMPL    MODE, #3
                                            02  14 00011         BGTR    1$
                                            52  D6 00013         INCL    R2
                                            51  D4 00015 1$:     CLRL    R1
                                            50  D5 00017         TSTL    MODE
                                            02  19 00019         BLSS    2$
                                            51  D6 0001B         INCL    R1
                              53            52  D2 0001D 2$:     MCOML   R2, R3
                              51            53  CA 00020         BICL2   R3, R1
                              01            51  D1 00023         CMPL    R1, #1
                                            63  12 00026         BNEQ    6$
                    51        10    AC      02  C7 00028         DIVL3   #2, INDEX, R1                  ; 0903
                              51            04  C0 0002D         ADDL2   #4, OFFSET
                              52        14  AC  D0 00030         MOVL    OPCODE, R2
                        FD    8F            52  91 00034         CMPB    R2, #253
                                            0E  13 00038         BEQL    3$
                              52          6142  7E 0003A         MOVAQ   (OFFSET)[R2], R2
                              51 00000000GEF42  9E 0003E         MOVAB   LIB$GB_OPINFO1[R2], R1
                                            11  11 00046         BRB     4$
                    52        52        F8  8F  78 00048 3$:     ASHL    #-8, R2, R2
                              51          6142  7E 0004D         MOVAQ   (OFFSET)[R2], R1
                              51 00000000GEF41  9E 00051         MOVAB   LIB$GB_OPINFO2[R1], R1
        53        10    AC      01            00  EF 00059 4$:     EXTZV   #0, #1, INDEX, R3
                              53            04  C4 0005F         MULL2   #4, R3
                              53            03  C0 00062         ADDL2   #3, R3
        52                61  01            53  EF 00065         EXTZV   R3, #1, (R1), R2
                              01            52  D1 0006A         CMPL    R2, #1
                                            04  13 0006D         BEQL    5$
                              50            02  D0 0006F         MOVL    #2, R0                          ; 0905
                                            04  00072         RET
        50                50  02            00  EF 00073 5$:     EXTZV   #0, #2, MODE, R0                ; 0907
                              50            10  C4 00078         MULL2   #16, R0
                    08    BC  50            54  C9 0007B         BISL3   REG, R0, @DISPL
                        04  BC  01            01  D0 00080         MOVL    #1, @FLAG                       ; 0908
                        0C  BC  01            01  D0 00084         MOVL    #1, @PTR_DISP_SIZE              ; 0909
                                          00C2  31  00088         BRW     23$                             ; 0910
                                            52  D4 0008B 6$:     CLRL    R2                              ; 0912
                              0F            50  D1 0008D         CMPL    MODE, #15
                                            02  14 00090         BGTR    7$
                                            52  D6 00092         INCL    R2
                                            51  D4 00094 7$:     CLRL    R1
                              0A            50  D1 00096         CMPL    MODE, #10
                                            02  19 00099         BLSS    8$
                                            51  D6 0009B         INCL    R1
                              53            52  D2 0009D 8$:     MCOML   R2, R3
                              51            53  CA 000A0         BICL2   R3, R1
```

```
                         01         51 D1 000A3         CMPL    R1, #1                              ;
                                    32 12 000A6         BNEQ    14$                                 ;
                     04  BC         01 CE 000A8         MNEGL   #1, @FLAG                           ; 0914
                05       0A         50 CF 000AC         CASEL   MODE, #10, #5                       ; 0916
    000C        000C     0016     0016    000B0 9$:     .WORD   12$-9$,-                            ;
                         0011     0011    000B8                 12$-9$,-                            ;
                                                                10$-9$,-                            ;
                                                                10$-9$,-                            ;
                                                                11$-9$,-                            ;
                                                                11$-9$                              ;
                         51         02 D0 000BC 10$:    MOVL    #2, R1                              ;
                                    08 11 000BF         BRB     13$                                 ;
                         51         04 D0 000C1 11$:    MOVL    #4, R1                              ;
                                    03 11 000C4         BRB     13$                                 ;
                         51         01 D0 000C6 12$:    MOVL    #1, R1                              ;
                     0C  BC         51 D0 000C9 13$:    MOVL    R1, @PTR_DISP_SIZE                  ;
                51   0C  BC         03 78 000CD         ASHL    #3, @PTR_DISP_SIZE, R1              ; 0923
    08   BC     6B       51         00 EE 000D2         EXTV    #0, R1, (STREAM_PTR), @DISPL        ;
                                    64 11 000D8         BRB     21$                                 ; 0924
                                    53 D4 000DA 14$:    CLRL    R3                                  ; 0927
                         0F         54 D1 000DC         CMPL    REG, #15                            ;
                                    02 12 000DF         BNEQ    15$                                 ;
                                    53 D6 000E1         INCL    R3                                  ;
                                    52 D4 000E3 15$:    CLRL    R2                                  ;
                         08         50 D1 000E5         CMPL    MODE, #8                            ;
                                    02 12 000E8         BNEQ    16$                                 ;
                                    52 D6 000EA         INCL    R2                                  ;
                                    51 D4 000EC 16$:    CLRL    R1                                  ;
                         09         50 D1 000EE         CMPL    MODE, #9                            ;
                                    02 12 000F1         BNEQ    17$                                 ;
                                    51 D6 000F3         INCL    R1                                  ;
                         51         52 C8 000F5 17$:    BISL2   R2, R1                              ;
                         55         53 D2 000F8         MCOML   R3, R5                              ;
                         51         55 CA 000FB         BICL2   R5, R1                              ;
                         01         51 D1 000FE         CMPL    R1, #1                              ;
                                    41 12 00101         BNEQ    22$                                 ;
                     04  BC         01 D0 00103         MOVL    #1, @FLAG                           ; 0929
                         09         50 D1 00107         CMPL    MODE, #9                            ; 0930
                                    06 12 0010A         BNEQ    18$                                 ;
                     0C  BC         04 D0 0010C         MOVL    #4, @PTR_DISP_SIZE                  ; 0932
                                    0D 11 00110         BRB     19$                                 ;
                         7E     10  AC 7D 00112 18$:    MOVQ    INDEX, -(SP)                        ; 0935
                  0000V  CF         02 FB 00116         CALLS   #2, INS_CONTEXT                     ;
                     0C  BC         50 D0 0011B         MOVL    R0, @PTR_DISP_SIZE                  ;
                         50     0C  BC D0 0011F 19$:    MOVL    @PTR_DISP_SIZE, R0                  ; 0936
                         04         50 D1 00123         CMPL    R0, #4                              ;
                                    03 15 00126         BLEQ    20$                                 ;
                         50         04 D0 00128         MOVL    #4, R0                              ;
                         50         08 C4 0012B 20$:    MULL2   #8, R0                              ;
    08   BC     6B       50         00 EF 0012E         EXTZV   #0, R0, (STREAM_PTR), @DISPL        ;
                         04     0C  BC D1 00134         CMPL    @PTR_DISP_SIZE, #4                  ; 0937
                                    04 15 00138         BLEQ    21$                                 ;
                         50         04 D0 0013A         MOVL    #4, R0                              ; 0939
                                       04 0013D         RET                                         ;
                         5B     0C  BC C0 0013E 21$:    ADDL2   @PTR_DISP_SIZE, STREAM_PTR          ; 0940
                                    09 11 00142         BRB     23$                                 ; 0941
                                04  BC D4 00144 22$:    CLRL    @FLAG                               ; 0945
```

```
                                    0C   BC  D4 00147          CLRL    @PTR_DISP_SIZE                          ; 0946
                                    08   BC  D4 0014A          CLRL    @DISPL                                  ; 0947
                               50        01  D0 0014D 23$:     MOVL    #1, R0                                  ; 0950
                                         04  00150            RET                                             ; 0954

; Routine Size:  337 bytes,    Routine Base:  Z$DEBUG_CODE + 0326
```

```
;  721          0955  1 %SBTTL 'INS_CONTEXT - Determine operand type'
;  722          0956  1 ROUTINE ins_context (index, opcode) =
;  723          0957  1
;  724          0958  1 !---
;  725          0959  1 !       This routine decides what context applies to the given
;  726          0960  1 !       operand for a specific opcode.  It is used because we need
;  727          0961  1 !       to know whether a pc-relative mode for this operand would
;  728          0962  1 !       require a byte, word, longword, or quadword operand.
;  729          0963  1 !
;  730          0964  1 ! Inputs:
;  731          0965  1 !
;  732          0966  1 !       index = Which operand we're dealing with.  This number
;  733          0967  1 !               must be 1, 2, ... 6.
;  734          0968  1 !       opcode = The opcode we are currently working on.
;  735          0969  1 !                (This parameter has already been validated.)
;  736          0970  1 !
;  737          0971  1 ! Routine value:
;  738          0972  1 !
;  739          0973  1 !       If some error is detected, we return false.  Otherwise we return
;  740          0974  1 !       the number of bytes from the instruction stream that the current
;  741          0975  1 !       operand reference should consume.
;  742          0976  1 !
;  743          0977  1 !   The value, 0 to 3, stored in the op_context field is simply
;  744          0978  1 !   our encoding of 4 values into a 2-bit field.  The 'number of
;  745          0979  1 !   bytes' entry, above, is the number we are actually after.
;  746          0980  1 !--
;  747          0981  1
;  748          0982  2 BEGIN
;  749          0983  2
;  750          0984  2 !
;  751          0985  2 ! check for any of the following error conditions:
;  752          0986  2 !       1) we don't recognize this opcode.
;  753          0987  2 !       2) we don't have enough information about it.
;  754          0988  2 !          (ie - it is reserved or yet to be defined).
;  755          0989  2 !       3) we know about it, but don't believe that it
;  756          0990  2 !          should have as many operands as what
;  757          0991  2 !          'index' implies.  this check is necessary
;  758          0992  2 !          because the 'nul' entry in the opinfo
;  759          0993  2 !          declaration macros results in the same value
;  760          0994  2 !          being encoded as the 'byt' ones do.  since
;  761          0995  2 !          we can cross-check for this error at this
;  762          0996  2 !          point (by looking at the op_numops entry for
;  763          0997  2 !          this opcode), it did not seem worth taking up more
;  764          0998  2 !          bits in the opinfo table to differentiate 'nul'
;  765          0999  2 !          and the others.
;  766          1000  2 !
;  767          1001  2
;  768          1002  2 IF .lib$gb_opinfo [.opcode, op_numops] EQL not_an_op
;  769          1003  2 THEN
;  770          1004  2     RETURN 0;                              ! Error 2, see above.
;  771          1005  2
;  772          1006  2 IF .lib$gb_opinfo [.opcode, op_numops] LSS .index OR .index LEQ 0
;  773          1007  2 THEN
;  774          1008  2     RETURN 0;                              ! Error 3, see above.
;  775          1009  2
;  776          1010  2 ! now it is just a matter of looking into our opinfo table
;  777          1011  2 ! where we get 0, 1, 2, or 3.  this just happens to be
```

```
;  778        1012  2 ! the power of 2 which we need to calculate the number
;  779        1013  2 ! of bytes occupied by the corresponding operand.
;  780        1014  2
;  781        1015  2 RETURN 1 ^ .lib$gb_opinfo [.opcode, op_context(.index)];
;  782        1016  2
;  783        1017  1 END;


                                        003C 00000 INS_CONTEXT:
                                                          .WORD   Save R2,R3,R4,R5                        ; 0956
                       55 00000000G  EF   9E 00002        MOVAB   LIB$GB_OPINFO1, R5                      ;
                       54 00000000G  EF   9E 00009        MOVAB   LIB$GB_OPINFO2, R4                      ;
                       50            04   D0 00010        MOVL    #4, OFFSET                              ; 1002
                       52       08   AC   D0 00013        MOVL    OPCODE, R2                              ;
                                     53   D4 00017        CLRL    R3                                      ;
                 FD    8F            52   91 00019        CMPB    R2, #253                                ;
                                     0C   13 0001D        BEQL    1$                                      ;
                                     53   D6 0001F        INCL    R3                                      ;
                       51          6042   7E 00021        MOVAQ   (OFFSET)[R2], R1                        ;
           50          51            55   C1 00025        ADDL3   R5, R1, R0                              ;
                                     0C   11 00029        BRB     2$                                      ;
           51          52       F8   8F   78 0002B 1$:    ASHL    #-8, R2, R1                             ;
                       50          6041   7E 00030        MOVAQ   (OFFSET)[R1], R0                        ;
                       50            54   C0 00034        ADDL2   R4, R0                                  ;
     0F    60          04            00   ED 00037 2$:    CMPZV   #0, #4, (R0), #15                       ;
                                     5D   13 0003C        BEQL    7$                                      ;
                       50            04   D0 0003E        MOVL    #4, OFFSET                              ; 1006
                       0A            53   E9 00041        BLBC    R3, 3$                                  ;
                       51          6042   7E 00044        MOVAQ   (OFFSET)[R2], R1                        ;
           50          51            55   C1 00048        ADDL3   R5, R1, R0                              ;
                                     0C   11 0004C        BRB     4$                                      ;
           51          52       F8   8F   78 0004E 3$:    ASHL    #-8, R2, R1                             ;
                       50          6041   7E 00053        MOVAQ   (OFFSET)[R1], R0                        ;
                       50            54   C0 00057        ADDL2   R4, R0                                  ;
04   AC    60          04            00   ED 0005A 4$:    CMPZV   #0, #4, (R0), INDEX                     ;
                                     39   19 00060        BLSS    7$                                      ;
                                04   AC   D5 00062        TSTL    INDEX                                   ;
                                     34   15 00065        BLEQ    7$                                      ;
           50    04    AC            02   C7 00067        DIVL3   #2, INDEX, R0                           ; 1015
                       50            04   C0 0006C        ADDL2   #4, OFFSET                              ;
                       0A            53   E9 0006F        BLBC    R3, 5$                                  ;
                       51          6042   7E 00072        MOVAQ   (OFFSET)[R2], R1                        ;
           50          51            55   C1 00076        ADDL3   R5, R1, R0                              ;
                                     0C   11 0007A        BRB     6$                                      ;
           51          52       F8   8F   78 0007C 5$:    ASHL    #-8, R2, R1                             ;
                       50          6041   7E 00081        MOVAQ   (OFFSET)[R1], R0                        ;
                       50            54   C0 00085        ADDL2   R4, R0                                  ;
     51    04    AC    01            00   EF 00088 6$:    EXTZV   #0, #1, INDEX, R1                       ;
                       51            04   C4 0008E        MULL2   #4, R1                                  ;
     52    60          03            51   EF 00091        EXTZV   R1, #3, (R0), R2                        ;
           50          01            52   78 00096        ASHL    R2, #1, R0                              ;
                                          04 0009A        RET                                             ;
                                     50   D4 0009B 7$:    CLRL    R0                                      ; 1017
                                          04 0009D        RET                                             ;
```

; Routine Size: 158 bytes,   Routine Base: Z$DEBUG_CODE + 0477

```
;   785          1018  1 %SBTTL 'PUT_REG - Print a register name'
;   786          1019  1 ROUTINE put_reg (reg, cs_flag): NOVALUE =
;   787          1020  1
;   788          1021  1 !---
;   789          1022  1 !       This routine takes 1 parameter which it assumes is
;   790          1023  1 !       the number of a vax register.  It then prints out
;   791          1024  1 !       'r' followed by the number (in decimal), unless the
;   792          1025  1 !       register number is 'special'.  These include:
;   793          1026  1 !
;   794          1027  1 !       Register number         Special name
;   795          1028  1 !
;   796          1029  1 !               12                 AP
;   797          1030  1 !               13                 FP
;   798          1031  1 !               14                 SP
;   799          1032  1 !               15                 PC
;   800          1033  1 !
;   801          1034  1 !       An additional parameter is used as a flag to indicate
;   802          1035  1 !       whether the register reference should be enclosed in
;   803          1036  1 !       round/square brackets or not.
;   804          1037  1 !
;   805          1038  1 ! Inputs:
;   806          1039  1 !
;   807          1040  1 !       reg = The register number.
;   808          1041  1 !       cs_flag = A flag to control printing before/after REG.
;   809          1042  1 !
;   810          1043  1 ! Outputs:
;   811          1044  1 !
;   812          1045  1 !       None.
;   813          1046  1 !--
;   814          1047  1
;   815          1048  2 BEGIN
;   816          1049  2
;   817          1050  2 LOCAL
;   818          1051  2     index;
;   819          1052  2
;   820          1053  2 BIND
;   821          1054  2     enclosing_cs = UPLIT BYTE('(',')','[',']'): VECTOR [,BYTE],
;   822          1055  2     regnames = UPLIT WORD('AP', 'FP', 'SP', 'PC'): VECTOR [,WORD];
;   823          1056  2
;   824          1057  2 !
;   825          1058  2 ! If we are to put out any enclosing strings,
;   826          1059  2 ! then we have been passed the INDEX which we
;   827          1060  2 ! need to pick this string out of the above
;   828          1061  2 ! vector.
;   829          1062  2 !
;   830          1063  2
;   831          1064  2 index = .cs_flag;
;   832          1065  2
;   833          1066  2 IF .index NEQ no_brackets
;   834          1067  2 THEN
;   835          1068  2     append_string(1, enclosing_cs [.index]);
;   836          1069  2
;   837          1070  2 ! Now print the actual register reference.
;   838          1071  2
;   839          1072  2 IF .reg LSS ap_reg
;   840          1073  2 THEN
;   841          1074  3     BEGIN
```

```
;  842          1075  3     append('R');
;  843          1076  3     append_decimal(.reg);
;  844          1077  3     END
;  845          1078  2 ELSE
;  846          1079  2     append_string(2, regnames[.reg-12]);
;  847          1080  2
;  848          1081  2 ! See again if there is any enclosing string.
;  849          1082  2
;  850          1083  2 IF .index NEQ no_brackets
;  851          1084  2 THEN
;  852          1085  2     append_string(1, enclosing_cs [.index+1]);
;  853          1086  2
;  854          1087  1 END;


                                                    28  00515 P.AAH:   .ASCII  \(\                                    ;
                                                    29  00516          .ASCII  \)\                                    ;
                                                    5B  00517          .ASCII  \[\                                    ;
                                                    5D  00518          .ASCII  \]\                                    ;
                                                        00519          .BLKB   1
                                                50  41  0051A P.AAI:   .ASCII  \AP\                                   ;
                                                50  46  0051C          .ASCII  \FP\                                   ;
                                                50  53  0051E          .ASCII  \SP\                                   ;
                                                43  50  00520          .ASCII  \PC\                                   ;
                                                    52  00522 P.AAJ:   .ASCII  \R\                                    ;

                                                               ENCLOSING_CS=       P.AAH
                                                               REGNAMES=           P.AAI


                                                          0FFC 00000 PUT_REG:.WORD   Save R2,R3,R4,R5,R6,R7,R8,R9,R10,R11      ; 1019
                                                    54      0000V CF  9E 00002         MOVAB   APPEND_STRING, R4               ;
                                                    52        08  AC  D0 00007         MOVL    CS_FLAG, INDEX                  ; 1064
                                                                  53  D4 0000B         CLRL    R3                              ; 1066
                                                    01                52  D1 0000D     CMPL    INDEX, #1                       ;
                                                                      0C  13 00010     BEQL    1$                              ;
                                                                      53  D6 00012     INCL    R3                              ;
                                                    51        DA AF42  9E 00014         MOVAB   ENCLOSING_CS[INDEX], R1         ; 1068
                                                    50                01  D0 00019     MOVL    #1, R0                          ;
                                                                      64  16 0001C     JSB     APPEND_STRING                   ;
                                                    0C          04    AC  D1 0001E 1$:  CMPL    REG, #T2                        ; 1072
                                                                      13  18 00022     BGEQ    2$                              ;
                                                    51        D8    AF  9E 00024     MOVAB   P.AAJ, R1                       ; 1075
                                                    50                01  D0 00028     MOVL    #1, R0                          ;
                                                                      64  16 0002B     JSB     APPEND_STRING                   ;
                                                                04    AC  DD 0002D     PUSHL   REG                             ; 1076
                                                0000V CF                01  FB 00030     CALLS   #1, APPEND_DECIMAL              ;
                                                                        0E  11 00035     BRB     3$                              ; 1072
                                                    50          04    AC  D0 00037 2$:  MOVL    REG, R0                         ; 1079
                                                    51        A0 AF40  3E 0003B         MOVAW   REGNAMES-24[R0], R1             ;
                                                    50                02  D0 00040     MOVL    #2, R0                          ;
                                                                      64  16 00043     JSB     APPEND_STRING                   ;
                                                    0A                53  E9 00045 3$:  BLBC    R3, 4$                          ; 1083
                                                    51        A7 AF42  9E 00048         MOVAB   ENCLOSING_CS+1[INDEX], R1       ; 1085
                                                    50                01  D0 0004D     MOVL    #1, R0                          ;
                                                                      64  16 00050     JSB     APPEND_STRING                   ;
```

```
                                        04 00052 4$:      RET                                                   ; 1087

; Routine Size:  83 bytes,    Routine Base:  Z$DEBUG_CODE + 0523
```

```
;  856          1088  1 %SBTTL 'APPEND_ADDRESS - Symbolize value and append it'
;  857          1089  1 ROUTINE append_address (value, absflag): NOVALUE =
;  858          1090  1
;  859          1091  1 !---
;  860          1092  1 !
;  861          1093  1 !       This routine converts a given absolute value to a symbol
;  862          1094  1 !       and an offset (if possible) and appends the resulting string
;  863          1095  1 !       to the current output buffer.
;  864          1096  1 !
;  865          1097  1 ! Inputs:
;  866          1098  1 !
;  867          1099  1 !       value = Absolute value to be converted
;  868          1100  1 !       absflag = True if absolute address, else relative address
;  869          1101  1 !
;  870          1102  1 ! Outputs:
;  871          1103  1 !
;  872          1104  1 !       Either the hex value or the symbol+offset is appended.
;  873          1105  1 !---
;  874          1106  1
;  875          1107  2 BEGIN
;  876          1108  2
;  877          1109  2 IF .user_symbolize_routine EQL 0
;  878          1110  2 THEN
;  879          1111  2     append_hex(.value,4)
;  880          1112  2 ELSE
;  881          1113  3     BEGIN
;  882          1114  3     LOCAL
;  883          1115  3         retlen: WORD,
;  884          1116  3         buffer_left: VECTOR [2];
;  885          1117  3     buffer_left [0] = .user_buffer_left;
;  886          1118  3     buffer_left [1] = .user_buffer_address;
;  887          1119  3     IF (.user_symbolize_routine)(value,buffer_left,retlen,absflag)
;  888          1120  3     THEN
;  889          1121  4         BEGIN
;  890          1122  4         user_buffer_address = .user_buffer_address + .retlen;
;  891          1123  4         user_buffer_left = .user_buffer_left - .retlen;
;  892          1124  4         END
;  893          1125  3     ELSE
;  894          1126  3         append_hex(.value,4);
;  895          1127  2     END;
;  896          1128  2
;  897          1129  1 END;


                                000C 00000 APPEND_ADDRESS:
                                           .WORD   Save R2,R3                                  ; 1089
                     53     FA8E   CF  9E 00002    MOVAB   USER_BUFFER_LEFT, R3                ;
                     5E            0C  C2 00007    SUBL2   #12, SP                             ;
                     52       F6   A3  D0 0000A    MOVL    USER_SYMBOLIZE_ROUTINE, R2          ; 1109
                                   26  13 0000E    BEQL    1$                                  ;
                04   AE            63  3C 00010    MOVZWL  USER_BUFFER_LEFT, BUFFER_LEFT       ; 1117
                08   AE       FA   A3  D0 00014    MOVL    USER_BUFFER_ADDRESS, BUFFER_LEFT+4  ; 1118
                              08   AC  9F 00019    PUSHAB  ABSFLAG                             ; 1119
                              04   AE  9F 0001C    PUSHAB  RETLEN                              ;
```

```
                                 0C   AE  9F 0001F          PUSHAB  BUFFER_LEFT                                    ;
                                 04   AC  9F 00022          PUSHAB  VALUE                                          ;
                              62      04  FB 00025          CALLS   #4, (R2)                                       ;
                              0B      50  E9 00028          BLBC    R0, 1$                                         ;
                              50      6E  3C 0002B          MOVZWL  RETLEN, R0                                     ; 1122
                           FA A3      50  C0 0002E          ADDL2   R0, USER_BUFFER_ADDRESS                        ;
                              63      6E  A2 00032          SUBW2   RETLEN, USER_BUFFER_LEFT                       ; 1123
                                      04  00035          RET                                                    ; 1119
                                      04  DD 00036 1$:      PUSHL   #4                                             ; 1126
                                 04   AC  DD 00038          PUSHL   VALUE                                          ;
                           0000V CF      02  FB 0003B          CALLS   #2, APPEND_HEX                                 ;
                                         04  00040          RET                                                    ; 1129

; Routine Size:  65 bytes,    Routine Base:  Z$DEBUG_CODE + 0576
```

```
;  899          1130  1 %SBTTL 'APPEND_HEX - Append variable size hex value'
;  900          1131  1 ROUTINE append_hex (value, bytes): NOVALUE =
;  901          1132  1
;  902          1133  1 !---
;  903          1134  1 !
;  904          1135  1 !       This routine appends a given hex value to the current output
;  905          1136  1 !       buffer.
;  906          1137  1 !
;  907          1138  1 ! Inputs:
;  908          1139  1 !
;  909          1140  1 !       value = Absolute value
;  910          1141  1 !       bytes = Number of bytes to display
;  911          1142  1 !
;  912          1143  1 ! Outputs:
;  913          1144  1 !
;  914          1145  1 !       The hex value is appended.
;  915          1146  1 !---
;  916          1147  1
;  917          1148  2 BEGIN
;  918          1149  2
;  919          1150  2 LOCAL
;  920          1151  2     number;
;  921          1152  2
;  922          1153  2 BIND
;  923          1154  2     digit_table = UPLIT BYTE('0123456789ABCDEF'): VECTOR [,BYTE];
;  924          1155  2
;  925          1156  2 number = .value;
;  926          1157  2
;  927          1158  2 IF .number LSS 0                          ! If negative value,
;  928          1159  2 THEN
;  929          1160  3     BEGIN
;  930          1161  3     append('-');                          ! Output minus sign
;  931          1162  3     number = -.number;                    ! and print the absolute value
;  932          1163  2     END;
;  933          1164  2
;  934          1165  2 DECR i FROM .bytes*8-4 TO 0 BY 4         ! For each nibble,
;  935          1166  2 DO
;  936          1167  2     append_string(1, digit_table [.number <.i,4>]); ! Output the digit
;  937          1168  2
;  938          1169  1 END;


45  44  43  42  41  39  38  37  36  35  34  33  32  31   30  005B7 P.AAK:  .ASCII  \0123456789ABCDEF\                          ;
                                                         46  005C6                                                          ;
                                                         2D  005C7 P.AAL:  .ASCII  \-\                                      ;

                                                                   DIGIT_TABLE=        P.AAK


                                                             0FFC 00000 APPEND_HEX:
                                                                        .WORD   Save R2,R3,R4,R5,R6,R7,R8,R9,R10,R11        ; 1131
                                                 53          04   AC  D0 00002          MOVL    VALUE, NUMBER              ; 1156
                                                                  0D  18 00006          BGEQ    1$                         ; 1158
                                                 51          F4   AF  9E 00008          MOVAB   P.AAL, R1                  ; 1161
                                                 50               01  D0 0000C          MOVL    #1, R0                     ;
                                                              0000V 30 0000F          BSBW    APPEND_STRING              ;
```

```
                         53              53  CE 00012             MNEGL   NUMBER, NUMBER                          ; 1162
                         52        08    AC  D0 00015 1$:         MOVL    BYTES, R2                               ; 1165
                         52              08  C4 00019             MULL2   #8, R2                                  ;
                                         10  11 0001C             BRB     3$                                      ;
          50             53              04  EF 0001E 2$:         EXTZV   I, #4, NUMBER, R0                       ; 1167
                         51        C8 AF40   9E 00023             MOVAB   DIGIT_TABLE[R0], R1                     ;
                         50              01  D0 00028             MOVL    #1, R0                                  ;
                                    0000V    30 0002B             BSBW    APPEND_STRING                           ;
                         52              04  C2 0002E 3$:         SUBL2   #4, I                                   ;
                                         EB  18 00031             BGEQ    2$                                      ;
                                             04 00033             RET                                             ; 1169

; Routine Size:  52 bytes,    Routine Base:  Z$DEBUG_CODE + 05C8
```

```
;  940          1170  1 %SBTTL 'APPEND_DECIMAL - Append unsigned decimal value'
;  941          1171  1 ROUTINE append_decimal (value): NOVALUE =
;  942          1172  1
;  943          1173  1 !---
;  944          1174  1 !
;  945          1175  1 !       This routine appends a given unsigned decimal value
;  946          1176  1 !       to the current output buffer.
;  947          1177  1 !
;  948          1178  1 ! Inputs:
;  949          1179  1 !
;  950          1180  1 !       value = Number to be output
;  951          1181  1 !
;  952          1182  1 ! Outputs:
;  953          1183  1 !
;  954          1184  1 !       The decimal value is appended, without any padding or fill.
;  955          1185  1 !---
;  956          1186  1
;  957          1187  2 BEGIN
;  958          1188  2
;  959          1189  2 LINKAGE
;  960          1190  2     recursive_jsb = JSB: GLOBAL(number=2);
;  961          1191  2
;  962          1192  2 GLOBAL REGISTER
;  963          1193  2     number = 2;
;  964          1194  2
;  965          1195  2 ROUTINE output_remaining_digits: recursive_jsb NOVALUE =
;  966          1196  3     BEGIN
;  967          1197  3     EXTERNAL REGISTER number=2;
;  968          1198  3     LOCAL char: BYTE;
;  969          1199  3     char = '0' + (.number MOD 10);
;  970          1200  3     number = .number / 10;
;  971          1201  3     IF .number NEQ 0 THEN output_remaining_digits();
;  972          1202  3     append_string(1, char);
;  973          1203  2     END;



                                        5E             04  C2 00000 OUTPUT_REMAINING_DIGITS:
                                                                   SUBL2   #4, SP                                  ; 1195
          7E            00              52             01  7A 00003         EMUL    #1, NUMBER, #0, -(SP)           ; 1199
          50            50              8E             0A  7B 00008         EDIV    #10, (SP)+, R0, R0              ;
                        6E              50             30  81 0000D         ADDB3   #48, R0, CHAR                   ;
                                        52             0A  C6 00011         DIVL2   #10, NUMBER                     ; 1200
                                                       02  13 00014         BEQL    1$                              ; 1201
                                                       E8  10 00016         BSBB    OUTPUT_REMAINING_DIGITS         ;
                                        51             6E  9E 00018 1$:     MOVAB   CHAR, R1                        ; 1202
                                        50             01  D0 0001B         MOVL    #1, R0                          ;
                                                  0000V 30  0001E         BSBW    APPEND_STRING                   ;
                                        5E             04  C0 00021         ADDL2   #4, SP                          ; 1203
                                                       05  00024         RSB                                     ;

; Routine Size:  37 bytes,    Routine Base:  Z$DEBUG_CODE + 05FC

;  974          1204  2
```

```
;  975          1205  2 number = .value;
;  976          1206  2 output_remaining_digits();
;  977          1207  2
;  978          1208  1 END;


                                        0FFC 00000 APPEND_DECIMAL:
                                                   .WORD   Save R2,R3,R4,R5,R6,R7,R8,R9,R10,R11        ; 1171
                                 52       04   AC  D0 00002          MOVL    VALUE, NUMBER                ; 1205
                                               D3  10 00006          BSBB    OUTPUT_REMAINING_DIGITS      ; 1206
                                                   04 00008          RET                                  ; 1208

; Routine Size:  9 bytes,    Routine Base:  Z$DEBUG_CODE + 0621
```

```
;  980          1209  1 %SBTTL 'APPEND_RAD50 - Append RAD50 characters'
;  981          1210  1 ROUTINE append_rad50 (nwords, words): NOVALUE =
;  982          1211  1
;  983          1212  1 !---
;  984          1213  1 !
;  985          1214  1 !       This routine converts a series of RAD50 words to ASCII and
;  986          1215  1 !       appends it to the current output buffer.
;  987          1216  1 !
;  988          1217  1 ! Inputs:
;  989          1218  1 !
;  990          1219  1 !       nwords = Number of words to convert
;  991          1220  1 !       words = Address of words to convert
;  992          1221  1 !
;  993          1222  1 ! Outputs:
;  994          1223  1 !
;  995          1224  1 !       The string is appended, without any padding or fill.
;  996          1225  1 !---
;  997          1226  1
;  998          1227  2 BEGIN
;  999          1228  2
; 1000          1229  2 MAP
; 1001          1230  2     words:      REF VECTOR [,WORD];     ! Address of array of words
; 1002          1231  2
; 1003          1232  2 LOCAL
; 1004          1233  2     number:     WORD,
; 1005          1234  2     char:       VECTOR [3,BYTE];        ! 3 character array
; 1006          1235  2
; 1007          1236  2 INCRU word_number FROM 0 TO .nwords-1   ! For each word to convert,
; 1008          1237  2 DO
; 1009          1238  3     BEGIN
; 1010          1239  3     number = .words [.word_number];     ! Get value of word
; 1011          1240  3
; 1012          1241  3     DECR i FROM 2 TO 0                  ! For 3 characters,
; 1013          1242  3     DO
; 1014          1243  4         BEGIN
; 1015          1244  4         char [.i] = .number MOD 40;     ! Get low order character
; 1016          1245  4         number = .number / 40;          ! and divide by 40
; 1017          1246  3         END;
; 1018          1247  3
; 1019          1248  3     INCR i FROM 0 TO 2                  ! For each of the 3 characters,
; 1020          1249  3     DO
; 1021          1250  4         BEGIN
; 1022          1251  4         SELECTONEU .char [.i]
; 1023          1252  4         OF
; 1024          1253  4             SET
; 1025          1254  4             [0]:        char [.i] = ' ';
; 1026          1255  4             [1 TO 26]:  char [.i] = .char [.i] + 'A' - 1;
; 1027          1256  4             [27]:       char [.i] = '$';
; 1028          1257  4             [OTHERWISE]: char [.i] = .char [.i] + '.' - 28;
; 1029          1258  4             TES;
; 1030          1259  4         append_string(1, char [.i]);
; 1031          1260  3         END;
; 1032          1261  2     END;
; 1033          1262  2
; 1034          1263  1 END;
```

```
                                 OFFC 00000 APPEND_RAD50:
                                            .WORD   Save R2,R3,R4,R5,R6,R7,R8,R9,R10,R11          ; 1210
                55        04    AC      01  C3 00002          SUBL3   #1, NWORDS, R5               ; 1236
                                        53  D4 00007          CLRL    WORD_NUMBER                  ; 1239
                                        58  10 00009          BSBB    8$                           ;
                          54      08 BC43  B0 0000B 1$:       MOVW    @WORDS[WORD_NUMBER], NUMBER  ;
                          50          02  D0 00010          MOVL    #2, I                        ; 1241
                          51          54  3C 00013 2$:       MOVZWL  NUMBER, R1                   ; 1244
             7E           0C          51  01  7A 00016          EMUL    #1, R1, #0, -(SP)            ;
             51           51          8E  28  7B 0001B          EDIV    #40, (SP)+, R1, R1           ;
                        6E40          51  90 00020          MOVB    R1, CHAR[I]                  ;
                          51          54  3C 00024          MOVZWL  NUMBER, R1                   ; 1245
                          51          28  C6 00027          DIVL2   #40, R1                      ;
                          54          51  B0 0002A          MOVW    R1, NUMBER                   ;
                          E3          50  F4 0002D          SOBGEQ  I, 2$                        ; 1241
                                      52  D4 00030          CLRL    I                            ; 1248
                52        52          5E  C1 00032 3$:       ADDL3   SP, I, R1                    ; 1251
                                      61  95 00036          TSTB    (R1)                         ; 1254
                                      05  12 00038          BNEQ    4$                           ;
                          61          20  90 0003A          MOVB    #32, (R1)                    ;
                                      18  11 0003D          BRB     7$                           ;
                          1A          61  91 0003F 4$:       CMPB    (R1), #26                    ; 1255
                                      06  1A 00042          BGTRU   5$                           ;
                          61      40  8F  80 00044          ADDB2   #64, (R1)                    ;
                                      0D  11 00048          BRB     7$                           ;
                          1B          61  91 0004A 5$:       CMPB    (R1), #27                    ; 1256
                                      05  12 0004D          BNEQ    6$                           ;
                          61          24  90 0004F          MOVB    #36, (R1)                    ;
                                      03  11 00C52          BRB     7$                           ;
                          61          12  80 00054 6$:       ADDB2   #18, (R1)                    ; 1257
                          50          01  D0 00057 7$:       MOVL    #1, R0                       ; 1259
                                  0000V   30 0005A          BSBW    APPEND_STRING                ;
                    D1    52          02  F3 0005D          AOBLEQ  #2, I, 3$                    ; 1248
                                      53  D6 00061          INCL    WORD_NUMBER                  ; 1236
                          55          53  D1 00063 8$:       CMPL    WORD_NUMBER, R5              ;
                                      A3  1B 00066          BLEQU   1$                           ;
                                          04 00068          RET                                  ; 1263

; Routine Size:  105 bytes,    Routine Base:  Z$DEBUG_CODE + 062A
```

```
;  1036          1264  1 %SBTTL 'APPEND_STRING - Append to output buffer'
;  1037          1265  1 ROUTINE append_string (length, string): append_linkage NOVALUE =
;  1038          1266  1
;  1039          1267  1 !----
;  1040          1268  1 !
;  1041          1269  1 !       Append a string to the current output buffer.
;  1042          1270  1 !
;  1043          1271  1 ! Inputs:
;  1044          1272  1 !
;  1045          1273  1 !       length = Length of string
;  1046          1274  1 !       string = Address of string
;  1047          1275  1 !
;  1048          1276  1 !       user_buffer_address = Address of next available byte in user buffer
;  1049          1277  1 !       user_buffer_left = Number of bytes left in user buffer
;  1050          1278  1 !
;  1051          1279  1 ! Outputs:
;  1052          1280  1 !
;  1053          1281  1 !       user_buffer_address, user_buffer_left are updated.
;  1054          1282  1 !----
;  1055          1283  1
;  1056          1284  2 BEGIN
;  1057          1285  2
;  1058          1286  2 IF .user_buffer_left GEQ .length        ! If enough room left,
;  1059          1287  2 THEN
;  1060          1288  3     BEGIN
;  1061          1289  3     CH$MOVE(.length, .string, .user_buffer_address);
;  1062          1290  3     user_buffer_address = .user_buffer_address + .length;
;  1063          1291  3     user_buffer_left = .user_buffer_left - .length;
;  1064          1292  2     END;
;  1065          1293  2
;  1066          1294  1 END;


                                        007C   8F   BB 00000 APPEND_STRING:
                                                             PUSHR   #^M<R2,R3,R4,R5,R6>                          ;  1265
                                  56            50   D0 00004          MOVL    R0, R6                               ;
                          56      F96A    CF    10   00   ED 00007          CMPZV   #0, #16, USER_BUFFER_LEFT, LENGTH    ;  1286
                                                     10   19 0000E          BLSS    1$                                   ;
                                  F95B    DF    61   56   28 00010          MOVC3   LENGTH, (STRING), @USER_BUFFER_ADDRESS ;  1289
                                          F956  CF   56   C0 00016          ADDL2   LENGTH, USER_BUFFER_ADDRESS          ;  1290
                                          F957  CF   56   A2 0001B          SUBW2   LENGTH, USER_BUFFER_LEFT             ;  1291
                                        007C   8F   BA 00020 1$:           POPR    #^M<R2,R3,R4,R5,R6>                  ;  1294
                                                     05 00024              RSB                                          ;

; Routine Size:  37 bytes,    Routine Base:  Z$DEBUG_CODE + 0693
```

```
; 1068          1295  1 END
; 1069          1296  0 ELUDOM



;                                   PSECT SUMMARY
;
;       Name                    Bytes                           Attributes
;
;  Z$DEBUG_CODE                    1720  NOVEC,  WRT,  RD ,  EXE,NOSHR,  LCL,  REL,  CON,  PIC,ALIGN(2)



;                                 Library Statistics
;
;                                          -------- Symbols --------      Pages       Processing
;       File                               Total    Loaded   Percent      Mapped      Time
;
;  _$255$DUA28:[SYSLIB]STARLET.L32;1        9776         3         0       581          00:00.8



;                                   COMMAND QUALIFIERS

;       BLISS/CHECK=(FIELD,INITIAL,OPTIMIZE)/LIS=LIS$:DECODE/OBJ=OBJ$:DECODE MSRC$:DECODE/UPDATE=(ENH$:DECODE)

; Size:         1655 code + 65 data bytes
; Run Time:        00:20.7
; Elapsed Time:    01:32.4
; Lines/CPU Min:    3754
; Lexemes/CPU-Min: 23351
; Memory Used:  165 pages
; Compilation Complete
```

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY