D 8

```
LLL                  IIIIIIIII     BBBBBBBBBBBB     RRRRRRRRRRRR     TTTTTTTTTTTTTTT  LLL
LLL                  IIIIIIIII     BBBBBBBBBBBB     RRRRRRRRRRRR     TTTTTTTTTTTTTTT  LLL
LLL                  IIIIIIIII     BBBBBBBBBBBB     RRRRRRRRRRRR     TTTTTTTTTTTTTTT  LLL
LLL                     III        BBB         BBB  RRR         RRR        TTT        LLL
LLL                     III        BBB         BBB  RRR         RRR        TTT        LLL
LLL                     III        BBB         BBB  RRR         RRR        TTT        LLL
LLL                     III        BBB         BBB  RRR         RRR        TTT        LLL
LLL                     III        BBB         BBB  RRR         RRR        TTT        LLL
LLL                     III        BBBBBBBBBBBB     RRRRRRRRRRRR           TTT        LLL
LLL                     III        BBBBBBBBBBBB     RRRRRRRRRRRR           TTT        LLL
LLL                     III        BBBBBBBBBBBB     RRRRRRRRRRRR           TTT        LLL
LLL                     III        BBB         BBB  RRR   RRR              TTT        LLL
LLL                     III        BBB         BBB  RRR   RRR              TTT        LLL
LLL                     III        BBB         BBB  RRR   RRR              TTT        LLL
LLL                     III        BBB         BBB  RRR      RRR           TTT        LLL
LLL                     III        BBB         BBB  RRR      RRR           TTT        LLL
LLL                     III        BBB         BBB  RRR      RRR           TTT        LLL
LLLLLLLLLLLLLLL      IIIIIIIII     BBBBBBBBBBBB     RRR         RRR        TTT        LLLLLLLLLLLLLLL
LLLLLLLLLLLLLLL      IIIIIIIII     BBBBBBBBBBBB     RRR         RRR        TTT        LLLLLLLLLLLLLLL
LLLLLLLLLLLLLLL      IIIIIIIII     BBBBBBBBBBBB     RRR         RRR        TTT        LLLLLLLLLLLLLLL
```

```
LL              IIIIII    BBBBBBBB    DDDDDDDD    EEEEEEEEEE    CCCCCCCC    OOOOOO    DDDDDDDD    FFFFFFFFFF
LL              IIIIII    BBBBBBBB    DDDDDDDD    EEEEEEEEEE    CCCCCCCC    OOOOOO    DDDDDDDD    FFFFFFFFFF
LL                II      BB      BB  DD      DD  EE          CC          OO      OO  DD      DD  FF
LL                II      BB      BB  DD      DD  EE          CC          OO      OO  DD      DD  FF
LL                II      BB      BB  DD      DD  EE          CC          OO      OO  DD      DD  FF
LL                II      BB      BB  DD      DD  EE          CC          OO      OO  DD      DD  FF
LL                II      BBBBBBBB    DD      DD  EEEEEEEE    CC          OO      OO  DD      DD  FFFFFFFF
LL                II      BBBBBBBB    DD      DD  EEEEEEEE    CC          OO      OO  DD      DD  FFFFFFFF
LL                II      BB      BB  DD      DD  EE          CC          OO      OO  DD      DD  FF
LL                II      BB      BB  DD      DD  EE          CC          OO      OO  DD      DD  FF
LL                II      BB      BB  DD      DD  EE          CC          OO      OO  DD      DD  FF              ....
LL                II      BB      BB  DD      DD  EE          CC          OO      OO  DD      DD  FF              ....
LLLLLLLLLL      IIIIII    BBBBBBBB    DDDDDDDD    EEEEEEEEEE    CCCCCCCC    OOOOOO    DDDDDDDD    FF              ....
LLLLLLLLLL      IIIIII    BBBBBBBB    DDDDDDDD    EEEEEEEEEE    CCCCCCCC    OOOOOO    DDDDDDDD    FF              ....


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
```

```
0000     1          .TITLE  LIB$DECODE_FAULT - Decode instruction stream
0000     2          .IDENT  /1-009/                 ; File: LIBDECODF.MAR Edit: SBL1009
0000     3
0000     4  ;*****************************************************************************
0000     5  ;*                                                                           *
0000     6  ;*  COPYRIGHT (c) 1978, 1980, 1982, 1984 BY                                  *
0000     7  ;*  DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.                   *
0000     8  ;*  ALL RIGHTS RESERVED.                                                     *
0000     9  ;*                                                                           *
0000    10  ;*  THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED     *
0000    11  ;*  ONLY IN  ACCORDANCE WITH  THE  TERMS  OF  SUCH  LICENSE  AND WITH THE    *
0000    12  ;*  INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR  ANY  OTHER     *
0000    13  ;*  COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY     *
0000    14  ;*  OTHER PERSON.  NO TITLE TO AND OWNERSHIP OF  THE  SOFTWARE IS  HEREBY     *
0000    15  ;*  TRANSFERRED.                                                             *
0000    16  ;*                                                                           *
0000    17  ;*  THE INFORMATION IN THIS SOFTWARE IS  SUBJECT TO CHANGE WITHOUT NOTICE     *
0000    18  ;*  AND  SHOULD  NOT  BE  CONSTRUED AS  A COMMITMENT BY DIGITAL EQUIPMENT    *
0000    19  ;*  CORPORATION.                                                             *
0000    20  ;*                                                                           *
0000    21  ;*  DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE  OR  RELIABILITY OF ITS     *
0000    22  ;*  SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.                  *
0000    23  ;*                                                                           *
0000    24  ;*                                                                           *
0000    25  ;*****************************************************************************
0000    26
0000    27  ;++
0000    28  ; FACILITY: General Utility Library
0000    29  ;
0000    30  ; ABSTRACT:
0000    31  ;
0000    32  ;       LIB$DECODE_FAULT is a procedure which analyzes the instruction
0000    33  ;       stream and environment at the time of an instruction fault and
0000    34  ;       which calls a user-supplied procedure to "handle" the fault.
0000    35  ;
0000    36  ; ENVIRONMENT: Runs at any access mode, AST Reentrant
0000    37  ;
0000    38  ; AUTHOR: Steven B. Lionel, 12-August-1981
0000    39  ;
0000    40  ;       NOTE:  This module contains a great amount of code adapted from
0000    41  ;              LIB$EMULATE, written by Derek Zave.  Because of the large
0000    42  ;              amount of common code between this module, LIB$EMULATE and
0000    43  ;              LIB$SIM_TRAP (also written by Derek Zave), all three
0000    44  ;              modules should be investigated if a problem should be found
0000    45  ;              in any one of them.
0000    46  ;
0000    47  ;
0000    48  ; MODIFIED BY:
0000    49  ;
0000    50  ; 1-001 - Original.  SBL 12-Aug-1981
0000    51  ; 1-002 - Make register change counters words instead of bytes, since the
0000    52  ;         modification can conceivably be greater than 256 bytes.
0000    53  ;         Increase user stack to 80 longwords to be safe.  SBL 11-Sept-1981
0000    54  ; 1-003 - Correct argument count test for user_arg and opcode_table.
0000    55  ;         Correct test for valid standard opcode.  Correct register-mode
0000    56  ;         operand processing.  SBL 20-Oct-1981
0000    57  ; 1-004 - Correct 1-byte vs. 2-byte opcode test. Swap order of Modify and
```

```
                     ;          Write access codes.  SBL 22-Oct-1981
                     ; 1-005 - Correct and rearrange order of exception-type checks.  Use
                     ;          LIB$GET_OPCODE if a BPT is found.  SBL 10-Dec-1981
                     ; 1-006 - If FPD is set, and the addressing mode is autoincrement or
                     ;          autoincrement deferred, and the base register is PC, do the
                     ;          autoincrement anyway.  This is because we must step the PC over
                     ;          the operand.  SBL 28-Jun-1982
                     ; 1-007 - Don't set PSL$V_TP if PSL$V_TBIT is set.  The architecture says
                     ;          we're not supposed to, and it gets in the way of doing trace
                     ;          trapping.  SBL 22-Nov-1982
                     ; 1-008 - Add new parameter original_registers to the action routine which
                     ;          contains the contents of the registers at the time of the fault
                     ;          BEFORE autoincrement/autodecrement.  Recognize new return status
                     ;          LIB$_RESTART from action routine to restart current instruction.
                     ;          This allows trace routines to be implemented. Retract 1-007, as
                     ;          the rest of the code properly manipulates TP.  SBL 11-May-1983
                     ; 1-009 - Fix "branch destination out of range".  SBL 24-May-1983
                     ;--
```

```
0000     77          .SBTTL  DECLARATIONS
0000     78 ;
0000     79 ; LIBRARY MACRO CALLS:
0000     80 ;
0000     81          $SSDEF                  ; System Status Codes
0000     82          $SFDEF                  ; Stack frame definitions
0000     83          $DSCDEF                 ; Descriptor codes
0000     84          $CHFDEF                 ; Condition handling codes
0000     85          $PSLDEF                 ; Processor Status Longword codes
0000     86          $LIBDCFDEF              ; LIB$DECODE_FAULT definitions
0000     87 ;
0000     88 ; EXTERNAL DECLARATIONS:
0000     89 ;
0000     90          .DSABL  GBL             ; Force all external symbols to be declared
0000     91          .EXTRN  SYS$CALL_HANDL  ; System routine that calls handlers
0000     92          .EXTRN  SYS$SRCHANDLER  ; System routine that looks for handlers
0000     93          .EXTRN  SYS$UNWIND      ; $UNWIND system service
0000     94          .EXTRN  LIB$GET_OPCODE  ; Get original opcode from debugger
0000     95          .EXTRN  LIB$STOP        ; Signal severe error
0000     96          .EXTRN  LIB$_INVARG     ; Invalid argument error code
0000     97          .EXTRN  LIB$_RESTART    ; Restart current instruction
0000     98 ;
0000     99 ; MACROS:
0000    100 ;
0000    101
0000    102 ;       Macro for Comparing Condition Codes
0000    103         .MACRO  CMPCOND COND,LOC
0000    104         CMPZV   #3,#26,LOC,#COND@-3
0000    105         .ENDM
0000    106
0000    107 ;       Macro for generating operand definition codes
0000    108         .MACRO  OPDEF   A1,A2,A3,A4,A5,A6
0000    109         .IRP    XCODE,<A1,A2,A3,A4,A5,A6>
0000    110         .IF NOT_BLANK XCODE
0000    111         .BYTE   <LIB$K_DCFTYP_%EXTRACT(1,1,XCODE)@LIB$V_DCFTYP>+ -
0000    112                 LIB$K_DCFACC_%EXTRACT(0,1,XCODE)
0000    113         .ENDC
0000    114         .ENDR
0000    115         .BYTE   LIB$K_DCFOPR_END
0000    116         .ENDM
0000    117
0000    118
0000    119 ;
0000    120 ; EQUATED SYMBOLS:
0000    121 ;
0000    122 ;       See body of routine
0000    123 ;
0000    124 ; OWN STORAGE:
0000    125 ;
0000    126 ;       NONE
0000    127 ;
0000    128 ; PSECT DECLARATIONS:
0000    129 ;
00000000    130         .PSECT _LIB$CODE PIC, USR, CON, REL, LCL, SHR, -
0000    131                         EXE, RD, NOWRT, LONG
0000    132
```

```
                     0000   134          ;  ****************************************************************
                     0000   135          ;  *                                                              *
                     0000   136          ;  *                                                              *
                     0000   137          ;  *                     Assorted Definitions                     *
                     0000   138          ;  *                                                              *
                     0000   139          ;  *                                                              *
                     0000   140          ;  ****************************************************************
                     0000   141          ;
                     0000   142          ;
                     0000   143          ;
                     0000   144          ;  Parameters
                     0000   145          ;
00000050             0000   146 CALL_ARGS =     80                    ; flexible stack space (longwords)
                     0000   147                                       ; This is enough for 16 octaword
                     0000   148                                       ; operands, plus some extra room.
                     0000   149          ;
                     0000   150          ;
                     0000   151          ;  Call Frame Layout
                     0000   152          ;
00000000             0000   153 HANDLER  =     0                     ; condition handler location
00000004             0000   154 SAVE_PSW =     4                     ; saved processor status word
00000006             0000   155 SAVE_MASK =    6                     ; register save mask
0000000E             0000   156 MASK_ALIGN =   14                    ; bit position of alignment bits
00000008             0000   157 SAVE_AP  =     8                     ; user's argument pointer
0000000C             0000   158 SAVE_FP  =     12                    ; user's frame pointer
00000010             0000   159 SAVE_PC  =     16                    ; return point
00000014             0000   160 REG_R0   =     20                    ; user's R0
00000018             0000   161 REG_R1   =     24                    ; user's R1
0000001C             0000   162 REG_R2   =     28                    ; user's R2
00000020             0000   163 REG_R3   =     32                    ; user's R3
00000024             0000   164 REG_R4   =     36                    ; user's R4
00000028             0000   165 REG_R5   =     40                    ; user's R5
0000002C             0000   166 REG_R6   =     44                    ; user's R6
00000030             0000   167 REG_R7   =     48                    ; user's R7
00000034             0000   168 REG_R8   =     52                    ; user's R8
00000038             0000   169 REG_R9   =     56                    ; user's R9
0000003C             0000   170 REG_R10  =     60                    ; user's R10
00000040             0000   171 REG_R11  =     64                    ; user's R11
00000044             0000   172 FRAME_END =    68                    ; end of call frame
                     0000   173          ;
                     0000   174          ;  Call Frame Extension Layout
                     0000   175          ;
00000044             0000   176 REG_AP   =     68                    ; user's AP
00000048             0000   177 REG_FP   =     72                    ; user's FP
0000004C             0000   178 REG_SP   =     76                    ; user's SP
00000050             0000   179 REG_PC   =     80                    ; user's PC
00000054             0000   180 PSL      =     84                    ; user's PSL
00000058             0000   181 LOCAL_END =    88                    ; end of our local storage
00000058             0000   182 TEMP     =     88                    ; temporary area for arithmetic
                     0000   183          ;
                     0000   184          ;  Local Storage Layout
                     0000   185          ;
FFFFFFFF             0000   186 SAVE_ALIGN =   HANDLER-1             ; saved copy of alignment bits
FFFFFFFE             0000   187 SAVE_PARCNT =  SAVE_ALIGN-1          ; saved copy of parameter count
FFFFFFFD             0000   188 MODE     =     SAVE_PARCNT-1         ; access mode for probes
FFFFFFFC             0000   189 FLAGS    =     MODE-1                ; indicator flag bits
FFFFFFF8             0000   190 SAVE_DEPTH =   FLAGS-4               ; saved copy of signal depth
```

```
FFFFFFF8  0000   191 SHORT_LOCAL =   SAVE_DEPTH              ; start of short local storage
FFFFFFF4  0000   192 ORIG_PC =       SHORT_LOCAL-4           ; original PC
FFFFFFF0  0000   193 ORIG_SP =       ORIG_PC-4               ; original SP
FFFFFFEC  0000   194 ORIG_FP =       ORIG_SP-4               ; original FP
FFFFFFE8  0000   195 ORIG_AP =       ORIG_FP-4               ; original AP
FFFFFFE4  0000   196 ORIG_R11 =      ORIG_AP-4               ; original R11
FFFFFFE0  0000   197 ORIG_R10 =      ORIG_R11-4              ; original R10
FFFFFFDC  0000   198 ORIG_R9 =       ORIG_R10-4              ; original R9
FFFFFFD8  0000   199 ORIG_R8 =       ORIG_R9-4               ; original R8
FFFFFFD4  0000   200 ORIG_R7 =       ORIG_R8-4               ; original R7
FFFFFFD0  0000   201 ORIG_R6 =       ORIG_R7-4               ; original R6
FFFFFFCC  0000   202 ORIG_R5 =       ORIG_R6-4               ; original R5
FFFFFFC8  0000   203 ORIG_R4 =       ORIG_R5-4               ; original R4
FFFFFFC4  0000   204 ORIG_R3 =       ORIG_R4-4               ; original R3
FFFFFFC0  0000   205 ORIG_R2 =       ORIG_R3-4               ; original R2
FFFFFFBC  0000   206 ORIG_R1 =       ORIG_R2-4               ; original R1
FFFFFFB8  0000   207 ORIG_R0 =       ORIG_R1-4               ; original R0
          0000   208
          0000   209 ;+
          0000   210 ; Define sixteen octawords to hold immediate mode
          0000   211 ; operands which are to be read.
          0000   212 ;-
          0000   213
FFFFFEB8  0000   214 READ_OPERANDS = ORIG_R0 - 256           ; 16 octawords for operands
          0000   215
          0000   216 ;+
          0000   217 ; Define array of read operand addresses to be passed to user's action
          0000   218 ; routine.
          0000   219 ;-
          0000   220
FFFFFE78  0000   221 READ_ADDRS =    READ_OPERANDS - 64      ; 16 longword addresses
          0000   222
          0000   223 ;+
          0000   224 ; Define array of write operand addresses to be passed to user's action
          0000   225 ; routine.
          0000   226 ;-
          0000   227
FFFFFE38  0000   228 WRITE_ADDRS =   READ_ADDRS - 64         ; 16 longword addresses
          0000   229
          0000   230
          0000   231 ;+
          0000   232 ; Define array of operand types to be passed to user's action routine.
          0000   233 ;-
          0000   234
FFFFFDF8  0000   235 OPERAND_TYPES = WRITE_ADDRS - 64        ; 16 longwords
          0000   236
FFFFFDF4  0000   237 N_OF_OPERANDS = OPERAND_TYPES- 4        ; Number of operands
FFFFFDF0  0000   238 INSTR_OPCODE =  N_OF_OPERANDS - 4       ; Zero-extended opcode
          0000   239
FFFFFDEC  0000   240 INSTR_DEF =     INSTR_OPCODE-4          ; address of instruction operand
          0000   241                                         ; definition table entry
FFFFFDE8  0000   242 USER_ACT_ARG =  INSTR_DEF-4             ; user action routine argument
FFFFFDE4  0000   243 USER_ACT_ENV =  USER_ACT_ARG-4          ; user action routine environment
FFFFFDE0  0000   244 USER_ACT_ADR =  USER_ACT_ENV-4          ; user action routine address
FFFFFDDC  0000   245 SAVE_SIGARGS =  USER_ACT_ADR-4          ; address of signal arguments
FFFFFDD8  0000   246 COND_NAME =     SAVE_SIGARGS-4          ; saved condition name
FFFFFDD8  0000   247 LOCAL_START =   COND_NAME               ; start of our local storage
```

```
                     0000   248          ;
                     0000   249          ;       Flag Bit Numbers
                     0000   250          ;
00000000             0000   251 V_REGISTER =    0                       ; Current operand is a register
00000001             0000   252 V_RESIGNAL =    1                       ; Resignal requested
                     0000   253          ;
                     0000   254          ;       Flag Bit Masks
                     0000   255          ;
00000001             0000   256 M_REGISTER =    1@V_REGISTER            ; Current operand is a register
00000002             0000   257 M_RESIGNAL =    1@V_RESIGNAL            ; Resignal requested
                     0000   258          ;
```

```
                     0000   260          .SBTTL  LIB$DECODE_FAULT - Decode instruction stream.
                     0000   261 ;++
                     0000   262 ; FUNCTIONAL DESCRIPTION:
                     0000   263 ;
                     0000   264 ;       This procedure is to be called by a condition handler at the
                     0000   265 ;       time of an instruction fault.  It determines the environment
                     0000   266 ;       of the fault, analyzes the instruction stream to locate
                     0000   267 ;       operands, and calls a user-supplied procedure to handle the
                     0000   268 ;       exception.
                     0000   269 ;
                     0000   270 ; CALLING SEQUENCE:
                     0000   271 ;
                     0000   272 ;       status.wlc.v = LIB$DECODE_FAULT (sigargs.rlu.ra, mechargs.rlu.ra,
                     0000   273 ;                                        user_action.cx.dp
                     0000   274 ;                                        [, user_arg.rz.v
                     0000   275 ;                                        [, opcode_table.rbu.ra]])
                     0000   276 ;
                     0000   277 ; FORMAL PARAMETERS:
                     0000   278 ;
00000004             0000   279          sigargs = 4             ; Address of signal arguments array
00000008             0000   280          mechargs = 8            ; Address of mechanism arguments array
0000000C             0000   281          user_action = 12        ; Address of descriptor of user-action
                     0000   282                                  ; procedure.  Datatype may be BPV, in
                     0000   283                                  ; which case the environment value is
                     0000   284                                  ; loaded into R1 before calling.  Other
                     0000   285                                  ; types are assumed to be ZEM.
00000010             0000   286          user_arg = 16           ; Argument to user action routine (optional)
00000014             0000   287          opcode_table = 20       ; Address of byte array that specifies
                     0000   288                                  ; additional opcode definitions.  See
                     0000   289                                  ; text below for more information.
                     0000   290 ;
                     0000   291 ; IMPLICIT INPUTS:
                     0000   292 ;
                     0000   293 ;       NONE
                     0000   294 ;
                     0000   295 ; IMPLICIT OUTPUTS:
                     0000   296 ;
                     0000   297 ;       NONE
                     0000   298 ;
                     0000   299 ; COMPLETION STATUS:
                     0000   300 ;
                     0000   301 ;       SS$_RESIGNAL    Resignal exception to next handler
                     0000   302 ;                       This status is returned if the current
                     0000   303 ;                       exception is not one of the recognized
                     0000   304 ;                       faults or if the instruction being
                     0000   305 ;                       executed can not be found in either the
                     0000   306 ;                       user-supplied instruction definition
                     0000   307 ;                       tables or our own.
                     0000   308 ;
                     0000   309 ; SIDE EFFECTS:
                     0000   310 ;
                     0000   311 ;       Stack frames are unwound back to the frame which generated
                     0000   312 ;       the exception.  Further side effects may be caused by the
                     0000   313 ;       user action routine.
                     0000   314 ;
                     0000   315 ;       LIB$_INVARG     Invalid argument to Run-Time Library
                     0000   316 ;                       This exception is signalled if an instruction
```

```
                                0000   317 ;                       definition in the user-supplied opcode
                                0000   318 ;                       table contains an invalid operand definition
                                0000   319 ;                       or if more than 16 operands are defined.
                                0000   320 ;                       Because this exception is signalled after the
                                0000   321 ;                       signal frames are unwound, the exception
                                0000   322 ;                       will appear to have come from a procedure called
                                0000   323 ;                       at the point of the faulting instruction.
                                0000   324 ;
                                0000   325 ;--
                                0000   326
                                0000   327          .ENTRY  LIB$DECODE_FAULT,-      ; entrance
                          003C  0000   328                  ^M<R2,R3,R4,R5>         ; entry mask
                    50  04 AC   D0     0002   329          MOVL    sigargs(AP),R0  ; R0 = signal array location
                    51  04 A0   D0     0006   330          MOVL    CHF$L_SIG_NAME(R0),R1 ; R1 = signal name
                                000A   331          CMPCOND SS$_ROPRAND,R1          ; Reserved operand fault?
                          5A    13     0013   332          BEQL    2$              ; Ok if it is
                                0015   333          CMPCOND SS$_FLTOVF_F,R1         ; Floating overflow fault?
                          4F    13     001E   334          BEQL    2$              ; Ok if it is
                                0020   335          CMPCOND SS$_FLTUND_F,R1         ; Floating underflow fault?
                          44    13     0029   336          BEQL    2$              ; Ok if it is
                                002B   337          CMPCOND SS$_FLTDIV_F,R1         ; Floating divide-by-zero fault?
                          39    13     0034   338          BEQL    2$              ; Ok if it is
                                0036   339          CMPCOND SS$_OPCDEC,R1           ; Opcode reserved to Digital?
                          2E    13     003F   340          BEQL    2$              ; Ok if it is
                                0041   341          CMPCOND SS$_OPCCUS,R1           ; Opcode reserved to customers and CSS?
                          23    13     004A   342          BEQL    2$              ; Ok if it is
                                004C   343          CMPCOND SS$_ACCVIO,R1           ; Access violation?
                          1C    13     0051   344          BEQL    2$              ; Ok if it is
                                0053   345          CMPCOND SS$_BREAK,R1            ; Breakpoint fault?
                          11    13     005C   346          BEQL    2$              ; Ok if it is
                                005E   347          CMPCOND SS$_TBIT,R1             ; Trace pending fault?
                          06    13     0067   348          BEQL    2$              ; Ok if it is
                50  0918 8F   3C     0069   349 1$:       MOVZWL  #SS$_RESIGNAL, R0       ; Resignal exception
                          04    006E   350          RET                             ; Return to caller
                                006F   351
                                006F   352 ;+
                                006F   353 ;  Check to see if we can at least read the instruction opcode.  If not,
                                006F   354 ;  simply resignal.  Get opcode into R3
                                006F   355 ;-
                                006F   356
                      51  60    D0     006F   357 2$:       MOVL    (R0),R1                 ; Get signal argument count
                51    FC A041   DE     0072   358           MOVAL   -4(R0)[R1],R1           ; Get address of PC/PSL pair
                      51  61    7D     0077   359           MOVQ    (R1),R1                 ; Get PC/PSL in R1-R2
            52  52    02  18    EF     007A   360           EXTZV   #PSL$V_CURMOD,#PSL$S_CURMOD,R2,R2 ; Get current access mode
                61    01  52    0C     007F   361           PROBER  R2,#1,(R1)              ; Can we read first byte?
                          E4    13     0083   362           BEQL    1$                      ; If not, resignal
                      53  61    9A     0085   363           MOVZBL  (R1),R3                 ; Get first opcde byte
                  FD  8F  53    91     0088   364           CMPB    R3,#^XFD                ; 2-byte opcode?
                          09    1F     008C   365           BLSSU   3$                      ; skip if not
                61    02  52    0C     008E   366           PROBER  R2,#2,(R1)              ; Probe both bytes
                          D5    13     0092   367           BEQL    1$                      ; Skip if not accessible
                      53  61    3C     0094   368           MOVZWL  (R1),R3                 ; Get both opcode bytes
                                0097   369
                                0097   370 ;+
                                0097   371 ;  If the opcode is BPT, and if the exception is not SS$_BREAK, see
                                0097   372 ;  if the debugger has modified the instruction stream.
                                0097   373 ;-
```

```
                03   53   D1   0097   374 3$:     CMPL    R3,#^X03                        ; Is it a BPT?
                     34   12   009A   375         BNEQ    4$                              ; No, skip
           50    04   AC   D0   009C   376         MOVL    sigargs(AP),R0                  ; Get condition name
                           00A0   377         CMPCOND SS$_BREAK,CHF$L_SIG_NAME(R0) ; Is it SS$_BREAK?
                     24   12   00AA   378         BNEQ    4$                              ; No, skip
                54   51   D0   00AC   379         MOVL    R1,R4                           ; Save PC
                     51   DD   00AF   380         PUSHL   R1                              ; Push PC of instruction
     00000000'GF     01   FB   00B1   381         CALLS   #1,G^LIB$GET_OPCODE             ; Try to get original instruction
                     51   54   D0   00B8   382         MOVL    R4,R1                           ; Restore PC
                     53   50   D0   00BB   383         MOVL    R0,R3                           ; R3 has original opcode
              FD 8F   50   91   00BE   384         CMPB    R0,#^XFD                        ; 2-byte opcode?
                     0C   1F   00C2   385         BLSSU   4$                              ; skip if not
           61    02   52   0C   00C4   386         PROBER  R2,#2,(R1)                      ; Can we read both bytes?
                     9F   13   00C8   387         BEQL    1$                              ; Resignal if not
                     53   61   3C   00CA   388         MOVZWL  (R1),R3                         ; Get second byte
                     53   50   90   00CD   389         MOVB    R0,R3                           ; Get first byte
                                00D0   390
                                00D0   391 ;+
                                00D0   392 ;  See if the opcode is defined in either the user's opcode table or
                                00D0   393 ;  our own.
                                00D0   394 ;-
                     55   53   D0   00D0   395 4$:     MOVL    R3, R5                          ; Save "real" opcode
                     05   6C   91   00D3   396         CMPB    (AP),#<opcode_table/4>          ; opcode_table present?
                          27   1F   00D6   397         BLSSU   STD_OPCODE                      ; No
                54    14   AC   D0   00D8   398         MOVL    opcode_table(AP),R4             ; Get address of user opcode table
                          21   13   00DC   399         BEQL    STD_OPCODE                      ; If no table, look in standard tables
                     51   84   9A   00DE   400 10$:     MOVZBL  (R4)+,R1                        ; Get first byte from table
              FD 8F   51   91   00E1   401         CMPB    R1,#^XFD                        ; Is it a 2-byte opcode?
                          0D   1F   00E5   402         BLSSU   11$                             ; Skip if not
                51    FF   A4   3C   00E7   403         MOVZWL  -1(R4),R1                       ; Get two-byte code
                     54   D6   00EB   404         INCL    R4                              ; Update table pointer
            FFFF 8F   51   B1   00ED   405         CMPW    R1,#^XFFFF                      ; End of opcode definitions?
                          0B   13   00F2   406         BEQL    STD_OPCODE                      ; If so, search standard tables
                     53   51   B1   00F4   407 11$:     CMPW    R1,R3                           ; Is this the right opcode?
                          41   13   00F7   408         BEQL    INSTR_FOUND                     ; If so, we've got it!
                          84   95   00F9   409 12$:     TSTB    (R4)+                           ; Skip to next opcode
                          FC   13   00FB   410         BEQL    12$                             ; Defined by end byte of zero
                          DF   11   00FD   411         BRB     10$                             ; Look at next opcode
                                00FF   412
                                00FF   413 ;+
                                00FF   414 ;  Search our standard tables of opcode definitions.
                                00FF   415 ;-
                                00FF   416
                                00FF   417 STD_OPCODE:
                   FD 8F   53   91   00FF   418         CMPB    R3,#^XFD                        ; Known two-byte "stick"?
                          14   1E   0103   419         BGEQU   20$                             ; Skip if maybe
     54    00000446'EF43   3C   0105   420         MOVZWL  TAB_1BYTE[R3],R4                ; Get pattern offset
                          25   13   010D   421         BEQL    NOT_FOUND                       ; If zero, unknown opcode
     54    00000446'EF44   9E   010F   422         MOVAB   TAB_1BYTE[R4],R4                ; Get pattern address
                          21   11   0117   423         BRB     INSTR_FOUND                     ; Pattern address in R4
                          19   1A   0119   424 20$:     BGTRU   NOT_FOUND                       ; We have only the FD stick
        53    53   F8 8F   78   011B   425         ASHL    #-8,R3,R3                       ; Get second byte alone
     54    00000646'EF43   3C   0120   426         MOVZWL  TAB_2BYTE[R3],R4                ; Get pattern address
                          0A   13   0128   427         BEQL    NOT_FOUND                       ; If zero, unknown opcode
     54    00000646'EF44   9E   012A   428         MOVAB   TAB_2BYTE[R4],R4                ; Get address of pattern
                          06   11   0132   429         BRB     INSTR_FOUND
                                0134   430
```

```
                     0134   431 ;+
                     0134   432 ; Come here if we can't find a definition for the instruction.
                     0134   433 ;-
                     0134   434
                     0134   435 NOT_FOUND:
          50 0918 8F 3C  0134   436         MOVZWL  #SS$_RESIGNAL,R0        ; resignal current exception
             04        0139   437         RET                             ; Return to CHF
                     013A   438
                     013A   439 ;+
                     013A   440 ;  We now know that we want to handle the exception.  Unwind the
                     013A   441 ;  stack frames back to the one which caused the exceptions.  We actually
                     013A   442 ;  don't reset SP until the very end.
                     013A   443 ;-
                     013A   444
                     013A   445 INSTR_FOUND:
       5E   F4 AD 9E 013A   446         MOVAB   -12(FP),SP              ; allocate stack space AP, FP, SP
    0FC0 8F    BB 013E   447         PUSHR   #^M<R6,R7,R8,R9,R10,R11> ; save registers R6-R11
       5E   18 C2 0142   448         SUBL2   #24,SP                  ; allocate space for R0,R1,R2,R3,R4,R5
       50   5D D0 0145   449         MOVL    FP,R0                   ; R0 = current frame pointer
                     0148   450
            60 D5 0148   451 4$:     TSTL    (R0)                    ; Does that frame have a handler?
            27 13 014A   452         BEQL    5$                      ; Skip if not
                     014C   453 ;+
                     014C   454 ;  Call frame's handler with SS$_UNWIND.  Note that this is not exactly
                     014C   455 ;  how SYS$UNWIND does it, but is our best approximation.  The difference
                     014C   456 ;  is that there is no protection from overlapping unwinds.
                     014C   457 ;-
          7E 50 D0 014C   458         MOVL    R0,-(SP)                ; Save our R0
             7E 7C 014F   459         CLRQ    -(SP)                   ; Construct mechanism argument list
                     0151   460                                         ; Use dummy R0-R1 since we'll
                     0151   461                                         ; ignore what the handler do to them
                     0151   462                                         ; anyway.
                7E D4 0151   463         CLRL    -(SP)                   ; Depth=0
                50 DD 0153   464         PUSHL   R0                      ; Establisher's FP
                04 DD 0155   465         PUSHL   #4                      ; Number of mechanism args
          7E 0920 8F 3C 0157   466         MOVZWL  #SS$_UNWIND, -(SP)      ; Create unwind signal argument list
                01 DD 015C   467         PUSHL   #1
             08 AE 9F 015E   468         PUSHAB  8(SP)                   ; Mechanism list location
             04 AE 9F 0161   469         PUSHAB  4(SP)                   ; Signal list location
          51   60 D0 0164   470         MOVL    (R0),R1                 ; Handler address
   00000000'GF 16 0167   471         JSB     G^SYS$CALL_HANDL        ; Call condition handler
          5E   24 C0 016D   472         ADDL2   #36,SP                  ; Pop back to our saved R0
          50   8E D0 0170   473         MOVL    (SP)+,R0                ; Restore our R0
                     0173   474 ;+
                     0173   475 ; Ok, we're back from calling the handler.  Now unwind the frame.
                     0173   476 ;-
51   06 A0   0C   00 EF 0173   477 5$:     EXTZV   #0,#12,SAVE_MASK(R0),R1 ; R1 = register save mask
          52   14 A0 9E 0179   478         MOVAB   REG_R0(R0),R2           ; R2 = start of registers in R0 frame
                53 D4 017D   479         CLRL    R3                      ; clear the register index
   53   51   0C   53 EA 017F   480 6$:     FFS     R3,#12,R1,R3            ; find the next saved register
                08 13 0184   481         BEQL    7$                      ; no more saved registers - bypass
             6E43   82 D0 0186   482         MOVL    (R2)+,(SP)[R3]          ; get the register value
                53 D6 018A   483         INCL    R3                      ; increment the register number
                F1 11 018C   484         BRB     6$                      ; look some more
      30 AE   08 A0 7D 018E   485 7$:     MOVQ    SAVE_AP(R0),48(SP)      ; get the values of AP and FP
10 A0   00000004'8F D1 0193   486         CMPL    #SYS$CALL_HANDL+4,16(R0); is this the condition handler ?
                06 13 0198   487         BEQL    8$                      ; yes - bypass
```

```
                50    34 AE    D0  019D   488          MOVL    52(SP),R0               ; R0 = location of next call frame
                               D0  11  01A1   489          BRB     5$                      ; unwind the frame
          50    04 AC    1C    C3  01A3   490 8$:       SUBL3   #28,sigargs(AP),R0      ; Find the address of the point
                                     01A8   491                                          ; before the signal arguments list
                                     01A8   492                                          ; where we will save the signal
                                     01A8   493                                          ; depth, user-action procedure and
                                     01A8   494                                          ; argument, operand pattern address
                                     01A8   495                                          ; and opcode.
             38 AE    50    D0  01A8   496          MOVL    R0,56(SP)               ; Set saved copy of SP
             51    08 AC    D0  01AC   497          MOVL    mechargs(AP),R1         ; R1 = mechanism array location
             6E    0C A1    7D  01B0   498          MOVQ    CHF$L_MCH_SAVR0(R1),(SP); get R0 and R1
          04 A0    08 A1    D0  01B4   499          MOVL    CHF$L_MCH_DEPTH(R1),4(R0) ; Save signal depth
             51    0C AC    D0  01B9   500          MOVL    user_action(AP),R1      ; Get user-action routine descriptor
             20    02 A1    91  01BD   501          CMPB    DSC$B_DTYPE(R1),#DSC$K_DTYPE_BPV ; Bound procedure value?
                            07  12  01C1   502          BNEQ    9$                      ; Skip if not
          08 A0    04 B1    7D  01C3   503          MOVQ    @DSC$A_POINTER(R1),8(R0); Fetch address and environment
                            08  11  01C8   504          BRB     10$                     ;
          08 A0    04 A1    D0  01CA   505 9$:       MOVL    DSC$A_POINTER(R1),8(R0) ; Fetch address
                   0C A0    D4  01CF   506          CLRL    12(R0)                  ; No environment
                   10 A0    7C  01D2   507 10$:      CLRQ    16(R0)                  ; Assume no user-arg
                      04    6C  91  01D5   508          CMPB    (AP),#<user_arg/4>      ; Is it present?
                            05  1F  01D8   509          BLSSU   11$                     ; No
          10 A0    10 AC    D0  01DA   510          MOVL    user_arg(AP),16(R0)     ; Fetch user-arg
             14 A0    54    D0  01DF   511 11$:      MOVL    R4,20(R0)               ; Save opcode pattern address
             18 A0    55    D0  01E3   512          MOVL    R5,24(R0)               ; Save opcode
             51    1C A0    D0  01E7   513          MOVL    28(R0),R1               ; Get signal argument list count
           60    1C A041    D0  01EB   514          MOVL    28(R0)[R1],(R0)         ; Move PSL to safe spot
           1C A041    50    D0  01F0   515          MOVL    R0,28(R0)[R1]           ; Move address of saved PSL to
                                     01F5   516                                          ; PSL's place on the stack
                                     01F5   517
                                     01F5   518 ;+
                                     01F5   519 ; At this point, the stack looks like this:
                                     01F5   520 ;
                                     01F5   521 ;                +-------------------------+ <-- SP at exception time
                                     01F5   522 ;                |    pointer "A"          |
                                     01F5   523 ;                +-------------------------+
                                     01F5   524 ;                |        PC               |
                                     01F5   525 ;                +-------------------------+
                                     01F5   526 ;                |   signal arg #n-2       | (if any)
                                     01F5   527 ;                +-------------------------+
                                     01F5   528 ;                | signal condition name   |
                                     01F5   529 ;                +-------------------------+
                                     01F5   530 ;                |  N of signal args       |
                                     01F5   531 ;                +-------------------------+
                                     01F5   532 ;                | "original" opcode       |
                                     01F5   533 ;                +-------------------------+
                                     01F5   534 ;                | operand pattern addr    |
                                     01F5   535 ;                +-------------------------+
                                     01F5   536 ;                | user-action argument    |
                                     01F5   537 ;                +-------------------------+
                                     01F5   538 ;                |user-action environment  |
                                     01F5   539 ;                +-------------------------+
                                     01F5   540 ;                | user-action address     |
                                     01F5   541 ;                +-------------------------+
                                     01F5   542 ;                |  signal depth           |
                                     01F5   543 ;                +-------------------------+
                                     01F5   544 ;                |      PSL                |  <- R0  <- pointer "A"
```

```
                     01F5   545 ;                +- - - - - - - - - - -+
                     01F5   546 ;                        ... (unknown locations)
                     01F5   547 ;                +- - - - - - - - - - -+
                     01F5   548 ;                |    saved SP          |
                     01F5   549 ;                +----------------------+
                     01F5   550 ;                |    saved FP          |
                     01F5   551 ;                +- - - - - - - - - - -+
                     01F5   552 ;                        ... (saved R2-AP)
                     01F5   553 ;                +- - - - - - - - - - -+
                     01F5   554 ;                |    saved R1          |
                     01F5   555 ;                +----------------------+
                     01F5   556 ;                |    saved R0          |  <- SP
                     01F5   557 ;                +----------------------+
                     01F5   558 ;
                     01F5   559 ;  We now restore registers R0-SP in a single POPR.  When we're done,
                     01F5   560 ;  SP will point to the saved PSL where R0 points now.
                     01F5   561 ;-
                     01F5   562
                7FFF 8F    BA   01F5   563          POPR    #^M<R0,R1,R2,R3,R4,R5,- ; restore registers R0-SP
                     01F9   564                          R6,R7,R8,R9,R10,R11,AP,FP,SP>
                     01F9   565
                     01F9   566 ;-
                     01F9   567 ; Compute distance we need to move SP to get CALL_ARGS arguments.
                     01F9   568 ;-
     7E  00000047 8F   1C AE   C3   01F9   569          SUBL3   28(SP),#<CALL_ARGS-9>,-(SP) ; Number of longwords to subtract
                     0202   570                                                          ; \\ 28(SP) is the number of sigargs
                     0202   571                                                          ; \\ The constant 9 includes 7
                     0202   572                                                          ; \\ other pushed arguments, 1 for
                     0202   573                                                          ; \\ the sigargs count and 1 for the
                     0202   574                                                          ; \\ longword which is pushed by
                     0202   575                                                          ; \\ this instruction.
                6E   04    C4   0202   576          MULL2   #4,(SP)                     ; Get number of bytes
                5E   6E    C2   0205   577          SUBL2   (SP),SP                     ; There are now CALL_ARGS longwords
                     0208   578                                                          ; from (SP) to where we've stored
                     0208   579                                                          ; "pointer A". The CALLS instruction
                     0208   580                                                          ; will push one more longword.
            11'AF 00000050 8F   FB   0208   581          CALLS   #CALL_ARGS,B^DECODE_FAULT       ; call the major routine
                     00     0210   582          HALT                                ; execution should never return here
                     0211   583          ;
```

```
0211    585          .SBTTL  DECODE_FAULT - major processing routine
0211    586          ;
0211    587          ;       DECODE_FAULT - major processing routine
0211    588          ;
0211    589          ;               parameters:     ( Described Below )
0211    590          ;
0211    591          ;       Discussion
0211    592          ;
0211    593          ;           This is the major processing routine for LIB$DECODE_FAULT.
0211    594          ; The parameter list consists of CALL_ARGS+1 longwords, the last
0211    595          ; several of which contain the signal arguments list (including
0211    596          ; PC and PSL), the handler depth (from the now-clobbered
0211    597          ; mechanism list) and the addresses of the user action routine,
0211    598          ; environment, argument, instruction operand pattern
0211    599          ; address, and instruction opcode.
0211    600          ;
0211    601          ;           When the routine is entered the CALLS instruction saves
0211    602          ; the user's registers R0 to R11 in order and saves AP and FP
0211    603          ; elsewhere in the frame. The routine extends the saved
0211    604          ; registers by saving the user's AP, FP, SP, PC, and PSL after
0211    605          ; the saved register area (the last two are taken from the
0211    606          ; parameter list).
0211    607          ;
0211    608          ;           Because we don't know the length of the signal argument
0211    609          ; list, we need a clue as to where the values we saved begin.
0211    610          ; This is found by looking at the very last argument.  There
0211    611          ; we saved the address into the list of where the remaining
0211    612          ; values start.  Refer to the previous page for more details.
0211    613          ;
0211    614          ;           The local storage is allocated by extending the stack
0211    615          ; The cell MODE is set equal to the current access mode for
0211    616          ; use in probing memory references. The alignment bits in the
0211    617          ; call frame and  the call parameter count are also saved so
0211    618          ; there is a safe copy to use when processing unwinds.  The
0211    619          ; original contents of the registers are saved.
0211    620          ;
0211    621          ;           The instruction PC is then loaded into R11 and
0211    622          ; the opcode saved.
0211    623          ; Next, each operand of the instruction is located.  For each
0211    624          ; operand, its type and location is stored in an array
0211    625          ; to be passed later to the user action routine.  However,
0211    626          ; if the FPD bit is set in the PSL, no operands are fetched.
0211    627          ; If this is the case, the number of operands passed to
0211    628          ; the user-action routine is zero.  However, the contents
0211    629          ; of register PC in the register array will point to the
0211    630          ; next instruction.  It is assumed that if FPD is set that
0211    631          ; the registers and/or stack contain preprocessed operands.
0211    632          ;
0211    633          ; Notes:  1. From the description of the way the simulated
0211    634          ;             register area is constructed, it is clear that
0211    635          ;             the length longword of the parameter list is
0211    636          ;             overwritten. All of the methods of leaving
0211    637          ;             put this longword back together. The
0211    638          ;             internal condition handler does this if it
0211    639          ;             detects an unwind.
0211    640          ;
0211    641          ;           2. The location of the instruction being processed is
```

```
                                0211   642          ;                       stored in the return PC for our frame
                                0211   643          ;                       so it may be easily located from the traceback
                                0211   644          ;                       report if we blow up.
                                0211   645          ;
                                0211   646 DECODE_FAULT:                                ; entrance
                           0FFF 0211   647                  .WORD   ^M<R0,R1,R2,R3,R4,R5,R6,R7,R8,R9,R10,R11> ; entry mask
                        5E   FDD8 CD    9E 0213   648          MOVAB   LOCAL_START(FP),SP      ; allocate the local storage
                50   06 AD   02   0E    EF 0218   649          EXTZV   #MASK_ALIGN,#2,SAVE_MASK(FP),R0 ; R0 = alignment bits
                      FF AD   50   90    90 021E   650          MOVB    R0,SAVE_ALIGN(FP)       ; save them
                    FE AD   50   8F    90 0222   651          MOVB    #CALL_ARGS,SAVE_PARCNT(FP) ; save the parameter count
                      FC AD        94    94 0227   652          CLRB    FLAGS(FP)               ; clear the flag bits
                    6D   0415'CF    9E 022A   653          MOVAB   W^COND_HANDLER,HANDLER(FP) ; set up the condition handler
                  44 AD    08 AD    7D 022F   654          MOVQ    SAVE_AP(FP),REG_AP(FP)  ; move user's AP and FP into place
                4C AD    0144 CC    9E 0234   655          MOVAB   4*<CALL_ARGS+1>(AP),REG_SP(FP) ; move user's SP into place
                50 AD    013C CC    D0 023A   656          MOVL    4*<CALL_ARGS-1>(AP),REG_PC(FP) ; move PC into place
                     50    14 AD    9E 0240   657          MOVAB   REG_R0(FP),R0           ; Address of current R0
                     51    B8 AD    9E 0244   658          MOVAB   ORIG_R0(FP),R1          ; Address to save original R0
                        81   80    7D 0248   659          MOVQ    (R0)+,(R1)+             ; Save R0-R1
                        81   80    7D 024B   660          MOVQ    (R0)+,(R1)+             ; Save R2-R3
                        81   80    7D 024E   661          MOVQ    (R0)+,(R1)+             ; Save R4-R5
                        81   80    7D 0251   662          MOVQ    (R0)+,(R1)+             ; Save R6-R7
                        81   80    7D 0254   663          MOVQ    (R0)+,(R1)+             ; Save R8-R9
                        81   80    7D 0257   664          MOVQ    (R0)+,(R1)+             ; Save R10-R11
                        81   80    7D 025A   665          MOVQ    (R0)+,(R1)+             ; Save AP-FP
                        81   80    7D 025D   666          MOVQ    (R0)+,(R1)+             ; Save SP-PC
                     50   0140 CC    D0 0260   667          MOVL    4*<CALL_ARGS>(AP),R0    ; Get value of "Pointer A"
                       54 AD    80    D0 0265   668          MOVL    (R0)+,PSL(FP)           ; Store user's PSL
                       F8 AD    80    D0 0269   669          MOVL    (R0)+,SAVE_DEPTH(FP)    ; Save handler depth
                     FDE0 CD    80    7D 026D   670          MOVQ    (R0)+,USER_ACT_ADR(FP)  ; Save user action routine address and
                                0272   671                                                 ; environment
                     FDE8 CD    80    D0 0272   672          MOVL    (R0)+,USER_ACT_ARG(FP)  ; Save user action argument
                          54    80    D0 0277   673          MOVL    (R0)+,R4                ; Get instruction pattern address
                     FDF0 CD    80    D0 027A   674          MOVL    (R0)+,INSTR_OPCODE(FP)  ; Get opcode
                          51    60    CE 027F   675          MNEGL   (R0),R1                 ; Get minus signal arg count
                     5E   FC AE41    DE 0282   676          MOVAL   -4(SP)[R1],SP           ; Allocate space for signal arguments
                                0287   677 ;+
                                0287   678 ; Copy signal arguments to safe place on stack.
                                0287   679 ;-
                          52    5E    D0 0287   680          MOVL    SP,R2                   ; R2 will step through the list
                          51    51    CE 028A   681          MNEGL   R1,R1                   ; Get positive signal arg count
                          82    80    D0 028D   682 1$:      MOVL    (R0)+,(R2)+             ; Move a longword of the sigargs
                             FA 51    F5 0290   683          SOBGTR  R1,1$                   ; Loop until done
                        62    54 AD    D0 0293   684          MOVL    PSL(FP),(R2)            ; Store PSL in list
                     FDDC CD    5E    D0 0297   685          MOVL    SP,SAVE_SIGARGS(FP)     ; Save address of signal arguments
                FDD8 CD    04 AE    D0 029C   686          MOVL    4(SP),COND_NAME(FP)     ; Save condition name
                                02A2   687
           50   54 AD   02   18    EF 02A2   688          EXTZV   #PSL$V_CURMOD,#PSL$S_CURMOD,PSL(FP),R0 ; Get current access mode
                      FD AD    50    90 02A8   689          MOVB    R0,MODE(FP)             ; save it for probes
                        5B    50 AD    D0 02AC   690          MOVL    REG_PC(FP),R11          ; R11 = location of instruction
                      10 AD    5B    D0 02B0   691          MOVL    R11,SAVE_PC(FP)         ; save it in the return PC
                FDF0 CD    FD 8F    91 02B4   692          CMPB    #^XFD,INSTR_OPCODE(FP)  ; Is it a 2-byte opcode?
                              05    1A 02BA   693          BGTRU   4$                      ; no - bypass
                           50 AD    D6 02BC   694          INCL    REG_PC(FP)              ; increment PC
                              5B    D6 02BF   695          INCL    R11                     ; R11 = location of next byte
                           50 AD    D6 02C1   696 4$:      INCL    REG_PC(FP)              ; increment PC
                                02C4   697
```

```
                        02C4    699 ;+
                        02C4    700 ; R4 now contains the address of the operand pattern for this instruction.
                        02C4    701 ; Set the TP bit in the PSL if T is on.  Then for each operand definition,
                        02C4    702 ; read the instruction stream to evaluate the operand.
                        02C4    703 ;-
                        02C4    704
        05 54 AD   04   E1  02C4    705         BBC     #PSL$V_TBIT,PSL(FP),5$  ; the trace enable bit is clear - skip
        00 54 AD   1E   E2  02C9    706         BBSS    #PSL$V_TP,PSL(FP),5$    ; set the trace enable bit
                   55   D4  02CE    707 5$:     CLRL    R5                      ; R5 keeps track of number of operands
                        02D0    708
                        02D0    709 OPERAND_LOOP:
                   56   84  9A  02D0    710         MOVZBL  (R4)+,R6                ; R6 = operand definition byte
                        32  13  02D3    711         BEQL    LAST_OPERAND            ; If so, we're done here
     58 56 03   00   EF  02D5    712         EXTZV   #LIB$V_DCFACC,#LIB$S_DCFACC,R6,R8 ; Access type into R8
                        1E  13  02DA    713         BEQL    INVALID_TYPE            ; Check for invalid type
                   06   58  D1  02DC    714         CMPL    R8,#LIB$K_DCFACC_B      ; Too large?
                        19  1A  02DF    715         BGTRU   INVALID_TYPE            ; If so, error
     59 56 05   03   EF  02E1    716         EXTZV   #LIB$V_DCFTYP,#LIB$S_DCFTYP,R6,R9 ; Data type into R9
                        12  13  02E6    717         BEQL    INVALID_TYPE            ; Check for invalid type
                   09   59  D1  02E8    718         CMPL    R9,#LIB$K_DCFTYP_H      ; Too large?
                        0D  1A  02EB    719         BGTRU   INVALID_TYPE            ; If so, error
           FDF8 CD45    56  9A  02ED    720         MOVZBL  R6,OPERAND_TYPES(FP)[R5]; Store operand type code
                  0763  30  02F3    721         BSBW    GET_SPECIFIER           ; Get the specifier
              D6 55     10  F2  02F6    722         AOBLSS  #16,R5,OPERAND_LOOP     ; Increment count of operands and loop
                        02FA    723 ;       BRB     INVALID_TYPE            ; Too many operand types
                        02FA    724
                        02FA    725 ;+
                        02FA    726 ; Come here if an access or data type code is invalid, or if there
                        02FA    727 ; are too many operands.
                        02FA    728 ;-
                        02FA    729
                        02FA    730 INVALID_TYPE:
     00000000'8F    DD  02FA    731         PUSHL   #LIB$_INVARG            ; Signal LIB$_INVARG
00000000'GF    01   FB  0300    732         CALLS   #1,G^LIB$STOP
                        0307    733
                        0307    734 ;+
                        0307    735 ; We now have all of the instruction operands found and their addresses and
                        0307    736 ; types stored.  Move the count of operands to N_OF_OPERANDS and call the
                        0307    737 ; user action routine.
                        0307    738 ;-
                        0307    739
                        0307    740 LAST_OPERAND:
     02 54 AD   1B   E1  0307    741         BBC     #PSL$V_FPD,PSL(FP),1$   ; Skip if not FPD
                   55   D4  030C    742         CLRL    R5                      ; No operands if FPD
      FDF4 CD    55   D0  030E    743 1$:     MOVL    R5,N_OF_OPERANDS(FP)
           B8 AD   9F  0313    744         PUSHAB  ORIG_R0(FP)             ; Address of original registers
         FDE8 CD   DD  0316    745         PUSHL   USER_ACT_ARG(FP)        ; User action argument
                   5D  DD  031A    746         PUSHL   FP                      ; "context" for signal routine
      00000FE8'EF   9F  031C    747         PUSHAB  USER_SIGNAL             ; Address of signal routine
         FDDC CD   DD  0322    748         PUSHL   SAVE_SIGARGS(FP)        ; Address of signal arguments list
         FE38 CD   9F  0326    749         PUSHAB  WRITE_ADDRS(FP)         ; Address of writeable operands
         FE78 CD   9F  032A    750         PUSHAB  READ_ADDRS(FP)          ; Address of readable operands
         FDF8 CD   9F  032E    751         PUSHAB  OPERAND_TYPES(FP)       ; Address of operand types
         FDF4 CD   9F  0332    752         PUSHAB  N_OF_OPERANDS(FP)       ; Address of number of operands
           14 AD   9F  0336    753         PUSHAB  REG_R0(FP)              ; Address of registers
           54 AD   9F  0339    754         PUSHAB  PSL(FP)                 ; Address of PSL
           10 AD   9F  033C    755         PUSHAB  SAVE_PC(FP)             ; Address of original PC
```

```
                FDF0 CD    9F   033F    756          PUSHAB  INSTR_OPCODE(FP)          ; Address of opcode
          50    FDE0 CD    7D   0343    757          MOVQ    USER_ACT_ADR(FP),R0       ; Get address/environment of routine
                60    0D   FB   0348    758          CALLS   #13,(R0)                  ; Call user's action routine
                  03 50    E8   034B    759          BLBS    R0,NORMAL_EXIT            ; Resume execution if success
                    0C86   31   034E    760          BRW     RESIGNAL                  ; Resignal if failure
                           0351    761          ;
```

```
                                0351   763          ;
                                0351   764          ;       NORMAL_EXIT - Normal End of Instruction Emulation
                                0351   765          ;
                                0351   766          ;               entered by branching
                                0351   767          ;
                                0351   768          ;               R0 contains return status from action routine
                                0351   769          ;
                                0351   770          ;       Discussion
                                0351   771          ;
                                0351   772          ;           This routine restores control to the user program whenever
                                0351   773          ;       the processing ends without causing an exception.
                                0351   774          ;       When we return, all of the registers, PC, and the PSL are
                                0351   775          ;       set to the emulated values.
                                0351   776          ;
                                0351   777          ;           The method of leaving consists of pushing the
                                0351   778          ;       user's PC and PSL onto the user's stack putting the saved AP
                                0351   779          ;       and FP back in their proper places in the frame and performing
                                0351   780          ;       the indicated adjustment so that when a RET instruction is
                                0351   781          ;       executed all of the registers up to FP will be restored and
                                0351   782          ;       the stack pointer will be positioned to the PC, PSL pair.
                                0351   783          ;
                                0351   784          ;
                                0351   785  NORMAL_EXIT:
00000000'8F    50    D1         0351   786          CMPL    R0, #LIB$_RESTART               ; entrance
                03    12         0358   787          BNEQ    10$                             ; Restart original instruction?
              0CE2    30         035A   788          BSBW    FAULT_RESET                     ; No, use REG_xx registers
          5E   F8 AD  9E         035D   789  10$:     MOVAB   SHORT_LOCAL(FP),SP              ; Restore registers
             50   08  D0         0361   790          MOVL    #8,R0                           ; shorten the frame
                  32  10         0364   791          BSBB    TEST_FRAME                      ; R0 = size of PC, PSL pair
          4C AD    08  C2         0366   792          SUBL2   #8,REG_SP(FP)                   ; make sure we have room to push it
       4C BD   50 AD  7D         036A   793          MOVQ    REG_PC(FP),@REG_SP(FP)          ; allocate room on the user's stack
       08 AD   44 AD  7D         036F   794          MOVQ    REG_AP(FP),SAVE_AP(FP)          ; push the PC, PSL pair
       10 AD   97'AF  9F         0374   795          MOVAB   B^1$,SAVE_PC(FP)                ; put the user's PC, PSL pair back
          50   48 AD  9E         0379   796          MOVAB   FRAME_END+4(FP),R0              ; store our return point
    51    4C AD   50  C3         037D   797          SUBL3   R0,REG_SP(FP),R1                ; R0 = location of end of frame
 52    51   02   00  EF         0382   798          EXTZV   #0,#2,R1,R2                     ; R1 = distance of user SP from it
06 AD   02   0E   52  F0         0387   799          INSV    R2,#MASK_ALIGN,#2,SAVE_MASK(FP) ; R2 = stack alignment
                50   52  C0         038D   800          ADDL2   R2,R0                           ; store it into the frame
  FC AD   51   FE  8F   78         0390   801          ASHL    #-2,R1,-4(R0)                   ; compute the parameter area location
                       04         0396   802          RET                                     ; store the parameter count
                       02         0397   803  1$:     REI                                     ; return (to next instruction)
                                  0398   804          ;                                       ; Return to the next user instruction
```

```
                                0398   806          ;
                                0398   807          ;       TEST_FRAME - Test Frame Location and Move If Necessary
                                0398   808          ;
                                0398   809          ;               entered by subroutine branching
                                0398   810          ;
                                0398   811          ;               parameter:      R0 = Size of Information to be Pushed
                                0398   812          ;
                                0398   813          ;               returns with    R0 = Distance Frame was Moved
                                0398   814          ;
                                0398   815          ;       Discussion
                                0398   816          ;
                                0398   817          ;           This routine determines whether or not the address given
                                0398   818          ;       by subtracting R0 from the user's stack pointer can be made
                                0398   819          ;       the location following a parameter list without the location
                                0398   820          ;       being within the local storage. If this cannot be done then
                                0398   821          ;       the entire procedure frame is moved so the condition can be
                                0398   822          ;       satisfied. The distance that the procedure frame was moved
                                0398   823          ;       is returned in R0. The value is zero if the frame is not
                                0398   824          ;       moved.
                                0398   825          ;
                                0398   826          ;       Note:   1. The switch from one frame location to another is
                                0398   827          ;                  performed by a single indivisible POPR instruction
                                0398   828          ;                  so we are never in an anomalous state.
                                0398   829          ;
                                0398   830          ;               2. If the frame is moved to a higher address, then
                                0398   831          ;                  the saved AP and FP are changed to the values of
                                0398   832          ;                  the emulated registers. The reason for this is
                                0398   833          ;                  that the move may overlay a valid frame so it is
                                0398   834          ;                  assumed that the user's AP and FP have been changed
                                0398   835          ;                  by the instruction to information about a new valid
                                0398   836          ;                  frame.
                                0398   837          ;
                                0398   838
                                0398   839 TEST_FRAME:                                    ; entrance
                          00    DD 0398   840               PUSHL   #0                    ; push a zero
50    4C AD    50    C3 039A   841               SUBL3   R0,REG_SP(FP),R0              ; compute pushed information address
                50    03 CA 039F   842               BICL2   #3,R0                 ; align the value
      51    58 AD    9E 03A2   843               MOVAB   LOCAL_END(FP),R1              ; R1 = end of local storage
            51    50 D1 03A6   844               CMPL    R0,R1                 ; does push extend below the frame ?
                  14 1E 03A9   845               BGEQU   2$                    ; no - bypass
   53       5E    03 CB 03AB   846               BICL3   #3,SP,R3              ; R3 = aligned stack pointer
            53    18 C2 03AF   847               SUBL2   #24,R3                ; adjust for additional pushes
            52    50 D0 03B2   848               MOVL    R0,R2                 ; R2 = address following moved frame
            53    52 D1 03B5   849               CMPL    R2,R3                 ; does it extend into the frame ?
                  18 1B 03B8   850               BLEQU   3$                    ; no - bypass
            52    53 D0 03BA   851               MOVL    R3,R2                 ; yes - use address below the frame
                  13 11 03BD   852               BRB     3$                    ; bypass
   52       0447 CD 9E 03BF   853 2$:           MOVAB   FRAME_END+1027(FP),R2         ; R2 = last possible parameter end
            52    50 D1 03C4   854               CMPL    R0,R2                 ; does the push end above it ?
                  49 1B 03C7   855               BLEQU   5$                    ; no - bypass
08 AD       44    AD 7D 03C9   856               MOVQ    REG_AP(FP),SAVE_AP(FP)        ; change the saved AP and FP
   52    50    03    CB 03CE   857               BICL3   #3,R0,R2              ; R2 = aligned user stack pointer
            52    51 C2 03D2   858 3$:           SUBL2   R1,R2                 ; R2 = distance of the move
                  52 DD 03D5   859               PUSHL   R2                    ; push the quantity
                6E42 9F 03D7   860               PUSHAB  (SP)[R2]              ; push the modified SP
                6D42 9F 03DA   861               PUSHAB  (FP)[R2]              ; push the modified FP
                6C42 9F 03DD   862               PUSHAB  (AP)[R2]              ; push the modified AP
```

```
                FF AD42   9F  03E0    863          PUSHAB  SAVE_ALIGN(FP)[R2]      ; push the new alignment bits location
                FE AD42   9F  03E4    864          PUSHAB  SAVE_PARCNT(FP)[R2]     ; push the new parameter count address
          53    48 AD42   9E  03E8    865          MOVAB   FRAME_END+4(FP)[R2],R3  ; R3 = new frame end + 4 location
          53    50   53    C3  03ED    866          SUBL3   R3,R0,R3                ; R3 = distance to user's SP
       7E   53     FE 8F    78  03F1    867          ASHL    #-2,R3,-(SP)            ; push the new parameter count
          50    51   5E    C3  03F6    868          SUBL3   SP,R1,R0                ; R0 = number of bytes to move
                51   5E    D0  03FA    869          MOVL    SP,R1                   ; R1 = location of bytes to move
                     52    D5  03FD    870          TSTL    R2                      ; will we extend the stack ?
                     03    18  03FF    871          BGEQ    4$                      ; no - skip
                5E   52    C0  0401    872          ADDL2   R2,SP                   ; yes - extend the stack pointer
           6142 61   50    28  0404    873 4$:      MOVC3   R0,(R1),(R1)[R2]        ; move the frame
                9E   8E    F6  0409    874          CVTLB   (SP)+,@(SP)+            ; store the new parameter count
                     9E    94  040C    875          CLRB    @(SP)+                  ; clear the new alignment bits
                7000 8F    BA  040E    876          POPR    #^M<AP,FP,SP>           ; switch to the new frame
                     01    BA  0412    877 5$:      POPR    #^M<R0>                 ; R0 = distance frame was moved
                           05  0414    878          RSB                             ; return
                               0415    879          ;
```

```
                                0415   881          ;
                                0415   882          ;       COND_HANDLER - Internal Condition Handler
                                0415   883          ;
                                0415   884          ;               parameters:     P1 = Signal Array Location
                                0415   885          ;                               P2 = Mechanism Array Location
                                0415   886          ;
                                0415   887          ;               returns with    R0 = Condition Response
                                0415   888          ;
                                0415   889          ;       Discussion
                                0415   890          ;
                                0415   891          ;           This routine is the internal condition handler for
                                0415   892          ;       LIB$DECODE_FAULT.  Since we don't make constructive use
                                0415   893          ;       of exceptions in its main procedure, this routine requests
                                0415   894          ;       resignaling of all conditions it intercepts.
                                0415   895          ;
                                0415   896          ;           If the condition is SS$_UNWIND which indicates that an
                                0415   897          ;       unwind is about to take place, then it restores the argument
                                0415   898          ;       count longword in the parameter list for the procedure so the
                                0415   899          ;       unwind will work properly.
                                0415   900          ;
                                0415   901 COND_HANDLER:                               ; entrance
                          0000  0415   902          .WORD   0                           ; entry mask
                     50   04 AC  7D   0417   903          MOVQ    4(AP),R0            ; R0,R1 = condition array locations
                                041B   904          CMPCOND SS$_UNWIND,4(R0)            ; is this an unwind ?
                          19   12   0425   905          BNEQ    1$                  ; no - bypass
                     50   04 A1  D0   0427   906          MOVL    4(R1),R0            ; R0 = frame location
                     51   FF A0  90   042B   907          MOVB    SAVE_ALIGN(R0),R1   ; R1 = safe copy of alignment bits
                     7E   FE AD  9A   042F   908          MOVZBL  SAVE_PARCNT(FP),-(SP) ; push the argument count
           06 A0  02   0E   51  F0   0433   909          INSV    R1,#MASK_ALIGN,#2,SAVE_MASK(R0) ; store align bits in frame
                     50   51  C0   0439   910          ADDL2   R1,R0               ; add to the frame location
                  44 A0   8E  D0   043C   911          MOVL    (SP)+,FRAME_END(R0) ; store the argument count
                50   0918 8F  3C   0440   912 1$:       MOVZWL  #SS$_RESIGNAL,R0    ; resignal the condition
                           04   0445   913          RET                             ; return
                                0446   914          ;
```

```
                     0446   916 ;+
                     0446   917 ; Instruction operand pattern tables.  There are two tables, one for
                     0446   918 ; 1-byte opcodes and the other for 2-byte opcodes.  Each entry has the
                     0446   919 ; offset from the beginning of the table to the pattern which describes
                     0446   920 ; that instruction.  If the offset is zero, no such instruction exists.
                     0446   921 ;-
                     0446   922
                     0446   923 TAB_1BYTE:
0400'                0446   924         .WORD   PATRN_HALT-TAB_1BYTE    ; 00    HALT
0400'                0448   925         .WORD   PATRN_NOP-TAB_1BYTE     ; 01    NOP
0400'                044A   926         .WORD   PATRN_REI-TAB_1BYTE     ; 02    REI
0400'                044C   927         .WORD   PATRN_BPT-TAB_1BYTE     ; 03    BPT
0400'                044E   928         .WORD   PATRN_RET-TAB_1BYTE     ; 04    RET
0400'                0450   929         .WORD   PATRN_RSB-TAB_1BYTE     ; 05    RSB
0400'                0452   930         .WORD   PATRN_LDPCTX-TAB_1BYTE  ; 06    LDPCTX
0400'                0454   931         .WORD   PATRN_SVPCTX-TAB_1BYTE  ; 07    SVPCTX
0401'                0456   932         .WORD   PATRN_CVTPS-TAB_1BYTE   ; 08    CVTPS
0401'                0458   933         .WORD   PATRN_CVTSP-TAB_1BYTE   ; 09    CVTSP
0406'                045A   934         .WORD   PATRN_INDEX-TAB_1BYTE   ; 0A    INDEX
040D'                045C   935         .WORD   PATRN_CRC-TAB_1BYTE     ; 0B    CRC
0412'                045E   936         .WORD   PATRN_PROBER-TAB_1BYTE  ; 0C    PROBER
0412'                0460   937         .WORD   PATRN_PROBEW-TAB_1BYTE  ; 0D    PROBEW
0416'                0462   938         .WORD   PATRN_INSQUE-TAB_1BYTE  ; 0E    INSQUE
0419'                0464   939         .WORD   PATRN_REMQUE-TAB_1BYTE  ; 0F    REMQUE
041C'                0466   940         .WORD   PATRN_BSBB-TAB_1BYTE    ; 10    BSBB
041C'                0468   941         .WORD   PATRN_BRB-TAB_1BYTE     ; 11    BRB
041C'                046A   942         .WORD   PATRN_BNEQ-TAB_1BYTE    ; 12    BNEQ,BNEQU
041C'                046C   943         .WORD   PATRN_BEQL-TAB_1BYTE    ; 13    BEQL,BEQLU
041C'                046E   944         .WORD   PATRN_BGTR-TAB_1BYTE    ; 14    BGTR
041C'                0470   945         .WORD   PATRN_BLEQ-TAB_1BYTE    ; 15    BLEQ
041E'                0472   946         .WORD   PATRN_JSB-TAB_1BYTE     ; 16    JSB
041E'                0474   947         .WORD   PATRN_JMP-TAB_1BYTE     ; 17    JMP
041C'                0476   948         .WORD   PATRN_BGEQ-TAB_1BYTE    ; 18    BGEQ
041C'                0478   949         .WORD   PATRN_BLSS-TAB_1BYTE    ; 19    BLSS
041C'                047A   950         .WORD   PATRN_BGTRU-TAB_1BYTE   ; 1A    BGTRU
041C'                047C   951         .WORD   PATRN_BLEQU-TAB_1BYTE   ; 1B    BLEQU
041C'                047E   952         .WORD   PATRN_BVC-TAB_1BYTE     ; 1C    BVC
041C'                0480   953         .WORD   PATRN_BVS-TAB_1BYTE     ; 1D    BVS
041C'                0482   954         .WORD   PATRN_BGEQU-TAB_1BYTE   ; 1E    BGEQU,BCC
041C'                0484   955         .WORD   PATRN_BLSSU-TAB_1BYTE   ; 1F    BLSSU,BCS
0420'                0486   956         .WORD   PATRN_ADDP4-TAB_1BYTE   ; 20    ADDP4
0425'                0488   957         .WORD   PATRN_ADDP6-TAB_1BYTE   ; 21    ADDP6
0420'                048A   958         .WORD   PATRN_SUBP4-TAB_1BYTE   ; 22    SUBP4
0425'                048C   959         .WORD   PATRN_SUBP6-TAB_1BYTE   ; 23    SUBP6
042C'                048E   960         .WORD   PATRN_CVTPT-TAB_1BYTE   ; 24    CVTPT
0425'                0490   961         .WORD   PATRN_MULP-TAB_1BYTE    ; 25    MULP
042C'                0492   962         .WORD   PATRN_CVTTP-TAB_1BYTE   ; 26    CVTTP
0425'                0494   963         .WORD   PATRN_DIVP-TAB_1BYTE    ; 27    DIVP
0432'                0496   964         .WORD   PATRN_MOVC3-TAB_1BYTE   ; 28    MOVC3
0432'                0498   965         .WORD   PATRN_CMPC3-TAB_1BYTE   ; 29    CMPC3
0436'                049A   966         .WORD   PATRN_SCANC-TAB_1BYTE   ; 2A    SCANC
0436'                049C   967         .WORD   PATRN_SPANC-TAB_1BYTE   ; 2B    SPANC
0438'                049E   968         .WORD   PATRN_MOVC5-TAB_1BYTE   ; 2C    MOVC5
0438'                04A0   969         .WORD   PATRN_CMPC5-TAB_1BYTE   ; 2D    CMPC5
0441'                04A2   970         .WORD   PATRN_MOVTC-TAB_1BYTE   ; 2E    MOVTC
0441'                04A4   971         .WORD   PATRN_MOVTUC-TAB_1BYTE  ; 2F    MOVTUC
0448'                04A6   972         .WORD   PATRN_BSBW-TAB_1BYTE    ; 30    BSBW
```

```
0448' 04A8   973          .WORD   PATRN_BRW-TAB_1BYTE     ; 31    BRW
044A' 04AA   974          .WORD   PATRN_CVTWL-TAB_1BYTE   ; 32    CVTWL
044D' 04AC   975          .WORD   PATRN_CVTWB-TAB_1BYTE   ; 33    CVTWB
0450' 04AE   976          .WORD   PATRN_MOVP-TAB_1BYTE    ; 34    MOVP
0450' 04B0   977          .WORD   PATRN_CMPP3-TAB_1BYTE   ; 35    CMPP3
0454' 04B2   978          .WORD   PATRN_CVTPL-TAB_1BYTE   ; 36    CVTPL
0420' 04B4   979          .WORD   PATRN_CMPP4-TAB_1BYTE   ; 37    CMPP4
0458' 04B6   980          .WORD   PATRN_EDITPC-TAB_1BYTE  ; 38    EDITPC
0420' 04B8   981          .WORD   PATRN_MATCHC-TAB_1BYTE  ; 39    MATCHC
045D' 04BA   982          .WORD   PATRN_LOCC-TAB_1BYTE    ; 3A    LOCC
045D' 04BC   983          .WORD   PATRN_SKPC-TAB_1BYTE    ; 3B    SKPC
044A' 04BE   984          .WORD   PATRN_MOVZWL-TAB_1BYTE  ; 3C    MOVZWL
0461' 04C0   985          .WORD   PATRN_ACBW-TAB_1BYTE    ; 3D    ACBW
0466' 04C2   986          .WORD   PATRN_MOVAW-TAB_1BYTE   ; 3E    MOVAW
0469' 04C4   987          .WORD   PATRN_PUSHAW-TAB_1BYTE  ; 3F    PUSHAW
046B' 04C6   988          .WORD   PATRN_ADDF2-TAB_1BYTE   ; 40    ADDF2
046E' 04C8   989          .WORD   PATRN_ADDF3-TAB_1BYTE   ; 41    ADDF3
046B' 04CA   990          .WORD   PATRN_SUBF2-TAB_1BYTE   ; 42    SUBF2
046E' 04CC   991          .WORD   PATRN_SUBF3-TAB_1BYTE   ; 43    SUBF3
046B' 04CE   992          .WORD   PATRN_MULF2-TAB_1BYTE   ; 44    MULF2
046E' 04D0   993          .WORD   PATRN_MULF3-TAB_1BYTE   ; 45    MULF3
046B' 04D2   994          .WORD   PATRN_DIVF2-TAB_1BYTE   ; 46    DIVF2
046E' 04D4   995          .WORD   PATRN_DIVF3-TAB_1BYTE   ; 47    DIVF3
0472' 04D6   996          .WORD   PATRN_CVTFB-TAB_1BYTE   ; 48    CVTFB
0475' 04D8   997          .WORD   PATRN_CVTFW-TAB_1BYTE   ; 49    CVTFW
0478' 04DA   998          .WORD   PATRN_CVTFL-TAB_1BYTE   ; 4A    CVTFL
0478' 04DC   999          .WORD   PATRN_CVTRFL-TAB_1BYTE  ; 4B    CVTRFL
047B' 04DE  1000          .WORD   PATRN_CVTBF-TAB_1BYTE   ; 4C    CVTBF
047E' 04E0  1001          .WORD   PATRN_CVTWF-TAB_1BYTE   ; 4D    CVTWF
0481' 04E2  1002          .WORD   PATRN_CVTLF-TAB_1BYTE   ; 4E    CVTLF
0484' 04E4  1003          .WORD   PATRN_ACBF-TAB_1BYTE    ; 4F    ACBF
0489' 04E6  1004          .WORD   PATRN_MOVF-TAB_1BYTE    ; 50    MOVF
048C' 04E8  1005          .WORD   PATRN_CMPF-TAB_1BYTE    ; 51    CMPF
0489' 04EA  1006          .WORD   PATRN_MNEGF-TAB_1BYTE   ; 52    MNEGF
048F' 04EC  1007          .WORD   PATRN_TSTF-TAB_1BYTE    ; 53    TSTF
0491' 04EE  1008          .WORD   PATRN_EMODF-TAB_1BYTE   ; 54    EMODF
0497' 04F0  1009          .WORD   PATRN_POLYF-TAB_1BYTE   ; 55    POLYF
049B' 04F2  1010          .WORD   PATRN_CVTFD-TAB_1BYTE   ; 56    CVTFD
0000  04F4  1011          .WORD   0                       ; 57
049E' 04F6  1012          .WORD   PATRN_ADAWI-TAB_1BYTE   ; 58    ADAWI
0000  04F8  1013          .WORD   0                       ; 59
0000  04FA  1014          .WORD   0                       ; 5A
0000  04FC  1015          .WORD   0                       ; 5B
04A1' 04FE  1016          .WORD   PATRN_INSQHI-TAB_1BYTE  ; 5C    INSQHI
04A1' 0500  1017          .WORD   PATRN_INSQTI-TAB_1BYTE  ; 5D    INSQTI
04A4' 0502  1018          .WORD   PATRN_REMQHI-TAB_1BYTE  ; 5E    REMQHI
04A4' 0504  1019          .WORD   PATRN_REMQTI-TAB_1BYTE  ; 5F    REMQTI
04A7' 0506  1020          .WORD   PATRN_ADDD2-TAB_1BYTE   ; 60    ADDD2
04AA' 0508  1021          .WORD   PATRN_ADDD3-TAB_1BYTE   ; 61    ADDD3
04A7' 050A  1022          .WORD   PATRN_SUBD2-TAB_1BYTE   ; 62    SUBD2
04AA' 050C  1023          .WORD   PATRN_SUBD3-TAB_1BYTE   ; 63    SUBD3
04A7' 050E  1024          .WORD   PATRN_MULD2-TAB_1BYTE   ; 64    MULD2
04AA' 0510  1025          .WORD   PATRN_MULD3-TAB_1BYTE   ; 65    MULD3
04A7' 0512  1026          .WORD   PATRN_DIVD2-TAB_1BYTE   ; 66    DIVD2
04AA' 0514  1027          .WORD   PATRN_DIVD3-TAB_1BYTE   ; 67    DIVD3
04AE' 0516  1028          .WORD   PATRN_CVTDB-TAB_1BYTE   ; 68    CVTDB
04B1' 0518  1029          .WORD   PATRN_CVTDW-TAB_1BYTE   ; 69    CVTDW
```

```
04B4' 051A  1030          .WORD   PATRN_CVTDL-TAB_1BYTE   ; 6A    CVTDL
04B4' 051C  1031          .WORD   PATRN_CVTRDL-TAB_1BYTE  ; 6B    CVTRDL
04B7' 051E  1032          .WORD   PATRN_CVTBD-TAB_TBYTE   ; 6C    CVTBD
04BA' 0520  1033          .WORD   PATRN_CVTWD-TAB_1BYTE   ; 6D    CVTWD
04BD' 0522  1034          .WORD   PATRN_CVTLD-TAB_1BYTE   ; 6E    CVTLD
04C0' 0524  1035          .WORD   PATRN_ACBD-TAB_TBYTE    ; 6F    ACBD
04C5' 0526  1036          .WORD   PATRN_MOVD-TAB_1BYTE    ; 70    MOVD
04C8' 0528  1037          .WORD   PATRN_CMPD-TAB_1BYTE    ; 71    CMPD
04C5' 052A  1038          .WORD   PATRN_MNEGD-TAB_1BYTE   ; 72    MNEGD
04CB' 052C  1039          .WORD   PATRN_TSTD-TAB_TBYTE    ; 73    TSTD
04CD' 052E  1040          .WORD   PATRN_EMODD-TAB_1BYTE   ; 74    EMODD
04D3' 0530  1041          .WORD   PATRN_POLYD-TAB_1BYTE   ; 75    POLYD
04D7' 0532  1042          .WORD   PATRN_CVTDF-TAB_1BYTE   ; 76    CVTDF
0000  0534  1043          .WORD   0                       ; 77
04DA' 0536  1044          .WORD   PATRN_ASHL-TAB_1BYTE    ; 78    ASHL
04DE' 0538  1045          .WORD   PATRN_ASHQ-TAB_1BYTE    ; 79    ASHQ
04E2' 053A  1046          .WORD   PATRN_EMUL-TAB_1BYTE    ; 7A    EMUL
04E7' 053C  1047          .WORD   PATRN_EDIV-TAB_1BYTE    ; 7B    EDIV
04EC' 053E  1048          .WORD   PATRN_CLRQ-TAB_1BYTE    ; 7C    CLRQ,CLRD,CLRG
04EE' 0540  1049          .WORD   PATRN_MOVQ-TAB_1BYTE    ; 7D    MOVQ
04F1' 0542  1050          .WORD   PATRN_MOVAQ-TAB_1BYTE   ; 7E    MOVAQ,MOVAD,MOVAG
04F4' 0544  1051          .WORD   PATRN_PUSHAQ-TAB_1BYTE  ; 7F    PUSHAQ,PUSHAD,PUSHAG
04F6' 0546  1052          .WORD   PATRN_ADDB2-TAB_TBYTE   ; 80    ADDB2
04F9' 0548  1053          .WORD   PATRN_ADDB3-TAB_1BYTE   ; 81    ADDB3
04F6' 054A  1054          .WORD   PATRN_SUBB2-TAB_1BYTE   ; 82    SUBB2
04F9' 054C  1055          .WORD   PATRN_SUBB3-TAB_1BYTE   ; 83    SUBB3
04F6' 054E  1056          .WORD   PATRN_MULB2-TAB_1BYTE   ; 84    MULB2
04F9' 0550  1057          .WORD   PATRN_MULB3-TAB_1BYTE   ; 85    MULB3
04F6' 0552  1058          .WORD   PATRN_DIVB2-TAB_1BYTE   ; 86    DIVB2
04F9' 0554  1059          .WORD   PATRN_DIVB3-TAB_1BYTE   ; 87    DIVB3
04F6' 0556  1060          .WORD   PATRN_BISB2-TAB_1BYTE   ; 88    BISB2
04F9' 0558  1061          .WORD   PATRN_BISB3-TAB_1BYTE   ; 89    BISB3
04F6' 055A  1062          .WORD   PATRN_BICB2-TAB_1BYTE   ; 8A    BICB2
04F9' 055C  1063          .WORD   PATRN_BICB3-TAB_1BYTE   ; 8B    BICB3
04F6' 055E  1064          .WORD   PATRN_XORB2-TAB_1BYTE   ; 8C    XORB2
04F9' 0560  1065          .WORD   PATRN_XORB3-TAB_1BYTE   ; 8D    XORB3
04FD' 0562  1066          .WORD   PATRN_MNEGB-TAB_1BYTE   ; 8E    MNEGB
0500' 0564  1067          .WORD   PATRN_CASEB-TAB_1BYTE   ; 8F    CASEB
04FD' 0566  1068          .WORD   PATRN_MOVB-TAB_TBYTE    ; 90    MOVB
0504' 0568  1069          .WORD   PATRN_CMPB-TAB_1BYTE    ; 91    CMPB
04FD' 056A  1070          .WORD   PATRN_MCOMB-TAB_1BYTE   ; 92    MCOMB
0504' 056C  1071          .WORD   PATRN_BITB-TAB_TBYTE    ; 93    BITB
0507' 056E  1072          .WORD   PATRN_CLRB-TAB_1BYTE    ; 94    CLRB
0509' 0570  1073          .WORD   PATRN_TSTB-TAB_1BYTE    ; 95    TSTB
050B' 0572  1074          .WORD   PATRN_INCB-TAB_1BYTE    ; 96    INCB
050B' 0574  1075          .WORD   PATRN_DECB-TAB_1BYTE    ; 97    DECB
050D' 0576  1076          .WORD   PATRN_CVTBL-TAB_1BYTE   ; 98    CVTBL
0510' 0578  1077          .WORD   PATRN_CVTBW-TAB_1BYTE   ; 99    CVTBW
050D' 057A  1078          .WORD   PATRN_MOVZBL-TAB_1BYTE  ; 9A    MOVZBL
0510' 057C  1079          .WORD   PATRN_MOVZBW-TAB_1BYTE  ; 9B    MOVZBW
0513' 057E  1080          .WORD   PATRN_ROTL-TAB_1BYTE    ; 9C    ROTL
0517' 0580  1081          .WORD   PATRN_ACBB-TAB_1BYTE    ; 9D    ACBB
051C' 0582  1082          .WORD   PATRN_MOVAB-TAB_1BYTE   ; 9E    MOVAB
041E' 0584  1083          .WORD   PATRN_PUSHAB-TAB_1BYTE  ; 9F    PUSHAB
049E' 0586  1084          .WORD   PATRN_ADDW2-TAB_TBYTE   ; A0    ADDW2
051F' 0588  1085          .WORD   PATRN_ADDW3-TAB_1BYTE   ; A1    ADDW3
049E' 058A  1086          .WORD   PATRN_SUBW2-TAB_1BYTE   ; A2    SUBW2
```

```
051F' 058C  1087          .WORD   PATRN_SUBW3-TAB_1BYTE   ; A3    SUBW3
049E' 058E  1088          .WORD   PATRN_MULW2-TAB_1BYTE   ; A4    MULW2
051F' 0590  1089          .WORD   PATRN_MULW3-TAB_1BYTE   ; A5    MULW3
049E' 0592  1090          .WORD   PATRN_DIVW2-TAB_1BYTE   ; A6    DIVW2
051F' 0594  1091          .WORD   PATRN_DIVW3-TAB_1BYTE   ; A7    DIVW3
049E' 0596  1092          .WORD   PATRN_BISW2-TAB_1BYTE   ; A8    BISW2
051F' 0598  1093          .WORD   PATRN_BISW3-TAB_1BYTE   ; A9    BISW3
049E' 059A  1094          .WORD   PATRN_BICW2-TAB_1BYTE   ; AA    BICW2
051F' 059C  1095          .WORD   PATRN_BICW3-TAB_1BYTE   ; AB    BICW3
049E' 059E  1096          .WORD   PATRN_XORW2-TAB_1BYTE   ; AC    XORW2
051F' 05A0  1097          .WORD   PATRN_XORW3-TAB_1BYTE   ; AD    XORW3
0523' 05A2  1098          .WORD   PATRN_MNEGW-TAB_1BYTE   ; AE    MNEGW
0526' 05A4  1099          .WORD   PATRN_CASEW-TAB_1BYTE   ; AF    CASEW
0523' 05A6  1100          .WORD   PATRN_MOVW-TAB_1BYTE    ; B0    MOVW
052A' 05A8  1101          .WORD   PATRN_CMPW-TAB_1BYTE    ; B1    CMPW
0523' 05AA  1102          .WORD   PATRN_MCOMW-TAB_1BYTE   ; B2    MCOMW
052A' 05AC  1103          .WORD   PATRN_BITW-TAB_1BYTE    ; B3    BITW
052D' 05AE  1104          .WORD   PATRN_CLRW-TAB_1BYTE    ; B4    CLRW
052F' 05B0  1105          .WORD   PATRN_TSTW-TAB_1BYTE    ; B5    TSTW
0531' 05B2  1106          .WORD   PATRN_INCW-TAB_1BYTE    ; B6    INCW
0531' 05B4  1107          .WORD   PATRN_DECW-TAB_1BYTE    ; B7    DECW
052F' 05B6  1108          .WORD   PATRN_BISPSW-TAB_1BYTE  ; B8    BISPSW
052F' 05B8  1109          .WORD   PATRN_BICPSW-TAB_1BYTE  ; B9    BICPSW
052F' 05BA  1110          .WORD   PATRN_POPR-TAB_1BYTE    ; BA    POPR
052F' 05BC  1111          .WORD   PATRN_PUSHR-TAB_1BYTE   ; BB    PUSHR
052F' 05BE  1112          .WORD   PATRN_CHMK-TAB_1BYTE    ; BC    CHMK
052F' 05C0  1113          .WORD   PATRN_CHME-TAB_1BYTE    ; BD    CHME
052F' 05C2  1114          .WORD   PATRN_CHMS-TAB_1BYTE    ; BE    CHMS
052F' 05C4  1115          .WORD   PATRN_CHMU-TAB_1BYTE    ; BF    CHMU
0533' 05C6  1116          .WORD   PATRN_ADDL2-TAB_1BYTE   ; C0    ADDL2
0536' 05C8  1117          .WORD   PATRN_ADDL3-TAB_1BYTE   ; C1    ADDL3
0533' 05CA  1118          .WORD   PATRN_SUBL2-TAB_1BYTE   ; C2    SUBL2
0536' 05CC  1119          .WORD   PATRN_SUBL3-TAB_1BYTE   ; C3    SUBL3
0533' 05CE  1120          .WORD   PATRN_MULL2-TAB_1BYTE   ; C4    MULL2
0536' 05D0  1121          .WORD   PATRN_MULL3-TAB_1BYTE   ; C5    MULL3
0533' 05D2  1122          .WORD   PATRN_DIVL2-TAB_1BYTE   ; C6    DIVL2
0536' 05D4  1123          .WORD   PATRN_DIVL3-TAB_1BYTE   ; C7    DIVL3
0533' 05D6  1124          .WORD   PATRN_BISL2-TAB_1BYTE   ; C8    BISL2
0536' 05D8  1125          .WORD   PATRN_BISL3-TAB_1BYTE   ; C9    BISL3
0533' 05DA  1126          .WORD   PATRN_BICL2-TAB_1BYTE   ; CA    BICL2
0536' 05DC  1127          .WORD   PATRN_BICL3-TAB_1BYTE   ; CB    BICL3
0533' 05DE  1128          .WORD   PATRN_XORL2-TAB_1BYTE   ; CC    XORL2
0536' 05E0  1129          .WORD   PATRN_XORL3-TAB_1BYTE   ; CD    XORL3
053A' 05E2  1130          .WORD   PATRN_MNEGL-TAB_1BYTE   ; CE    MNEGL
053D' 05E4  1131          .WORD   PATRN_CASEL-TAB_1BYTE   ; CF    CASEL
053A' 05E6  1132          .WORD   PATRN_MOVL-TAB_1BYTE    ; D0    MOVL
0541' 05E8  1133          .WORD   PATRN_CMPL-TAB_1BYTE    ; D1    CMPL
053A' 05EA  1134          .WORD   PATRN_MCOML-TAB_1BYTE   ; D2    MCOML
0541' 05EC  1135          .WORD   PATRN_BITL-TAB_1BYTE    ; D3    BITL
0544' 05EE  1136          .WORD   PATRN_CLRL-TAB_1BYTE    ; D4    CLRL,CLRF
0546' 05F0  1137          .WORD   PATRN_TSTL-TAB_1BYTE    ; D5    TSTL
0548' 05F2  1138          .WORD   PATRN_INCL-TAB_1BYTE    ; D6    INCL
0548' 05F4  1139          .WORD   PATRN_DECL-TAB_1BYTE    ; D7    DECL
0533' 05F6  1140          .WORD   PATRN_ADWC-TAB_1BYTE    ; D8    ADWC
0533' 05F8  1141          .WORD   PATRN_SBWC-TAB_1BYTE    ; D9    SBWC
0541' 05FA  1142          .WORD   PATRN_MTPR-TAB_1BYTE    ; DA    MTPR
053A' 05FC  1143          .WORD   PATRN_MFPR-TAB_1BYTE    ; DB    MFPR
```

```
0544' 05FE  1144          .WORD   PATRN_MOVPSL-TAB_1BYTE  ; DC    MOVPSL
0546' 0600  1145          .WORD   PATRN_PUSHL-TAB_1BYTE   ; DD    PUSHL
054A' 0602  1146          .WORD   PATRN_MOVAL-TAB_1BYTE   ; DE    MOVAL,MOVAF
054D' 0604  1147          .WORD   PATRN_PUSHAL-TAB_1BYTE  ; DF    PUSHAL,PUSHAF
054F' 0606  1148          .WORD   PATRN_BBS-TAB_1BYTE     ; E0    BBS
054F' 0608  1149          .WORD   PATRN_BBC-TAB_1BYTE     ; E1    BBC
054F' 060A  1150          .WORD   PATRN_BBSS-TAB_1BYTE    ; E2    BBSS
054F' 060C  1151          .WORD   PATRN_BBCS-TAB_1BYTE    ; E3    BBCS
054F' 060E  1152          .WORD   PATRN_BBSC-TAB_1BYTE    ; E4    BBSC
054F' 0610  1153          .WORD   PATRN_BBCC-TAB_1BYTE    ; E5    BBCC
054F' 0612  1154          .WORD   PATRN_BBSSI-TAB_1BYTE   ; E6    BBSSI
054F' 0614  1155          .WORD   PATRN_BBCCI-TAB_1BYTE   ; E7    BBCCI
0553' 0616  1156          .WORD   PATRN_BLBS-TAB_1BYTE    ; E8    BLBS
0553' 0618  1157          .WORD   PATRN_BLBC-TAB_1BYTE    ; E9    BLBC
0556' 061A  1158          .WORD   PATRN_FFS-TAB_1BYTE     ; EA    FFS
0556' 061C  1159          .WORD   PATRN_FFC-TAB_1BYTE     ; EB    FFC
055B' 061E  1160          .WORD   PATRN_CMPV-TAB_1BYTE    ; EC    CMPV
055B' 0620  1161          .WORD   PATRN_CMPZV-TAB_1BYTE   ; ED    CMPZV
0556' 0622  1162          .WORD   PATRN_EXTV-TAB_1BYTE    ; EE    EXTV
0556' 0624  1163          .WORD   PATRN_EXTZV-TAB_1BYTE   ; EF    EXTZV
0560' 0626  1164          .WORD   PATRN_INSV-TAB_1BYTE    ; F0    INSV
0565' 0628  1165          .WORD   PATRN_ACBL-TAB_1BYTE    ; F1    ACBL
056A' 062A  1166          .WORD   PATRN_AOBLSS-TAB_1BYTE  ; F2    AOBLSS
056A' 062C  1167          .WORD   PATRN_AOBLEQ-TAB_1BYTE  ; F3    AOBLEQ
056E' 062E  1168          .WORD   PATRN_SOBGEQ-TAB_1BYTE  ; F4    SOBGEQ
056E' 0630  1169          .WORD   PATRN_SOBGTR-TAB_1BYTE  ; F5    SOBGTR
0571' 0632  1170          .WORD   PATRN_CVTLB-TAB_1BYTE   ; F6    CVTLB
0574' 0634  1171          .WORD   PATRN_CVTLW-TAB_1BYTE   ; F7    CVTLW
0577' 0636  1172          .WORD   PATRN_ASHP-TAB_1BYTE    ; F8    ASHP
057E' 0638  1173          .WORD   PATRN_CVTLP-TAB_1BYTE   ; F9    CVTLP
0585' 063A  1174          .WORD   PATRN_CALLG-TAB_1BYTE   ; FA    CALLG
0582' 063C  1175          .WORD   PATRN_CALLS-TAB_1BYTE   ; FB    CALLS
0400' 063E  1176          .WORD   PATRN_XFC-TAB_1BYTE     ; FC    XFC
0000  0640  1177          .WORD   0                       ; FD    (2-byte opcode)
0000  0642  1178          .WORD   0                       ; FE    (2-byte opcode)
0000  0644  1179          .WORD   0                       ; FF    (2-byte opcode)
      0646  1180
      0646  1181  TAB_2BYTE:
0000  0646  1182          .WORD   0                       ; 00FD
0000  0648  1183          .WORD   0                       ; 01FD
0000  064A  1184          .WORD   0                       ; 02FD
0000  064C  1185          .WORD   0                       ; 03FD
0000  064E  1186          .WORD   0                       ; 04FD
0000  0650  1187          .WORD   0                       ; 05FD
0000  0652  1188          .WORD   0                       ; 06FD
0000  0654  1189          .WORD   0                       ; 07FD
0000  0656  1190          .WORD   0                       ; 08FD
0000  0658  1191          .WORD   0                       ; 09FD
0000  065A  1192          .WORD   0                       ; 0AFD
0000  065C  1193          .WORD   0                       ; 0BFD
0000  065E  1194          .WORD   0                       ; 0CFD
0000  0660  1195          .WORD   0                       ; 0DFD
0000  0662  1196          .WORD   0                       ; 0EFD
0000  0664  1197          .WORD   0                       ; 0FFD
0000  0666  1198          .WORD   0                       ; 10FD
0000  0668  1199          .WORD   0                       ; 11FD
0000  066A  1200          .WORD   0                       ; 12FD
```

```
0000  066C  1201          .WORD   0                               ; 13FD
0000  066E  1202          .WORD   0                               ; 14FD
0000  0670  1203          .WORD   0                               ; 15FD
0000  0672  1204          .WORD   0                               ; 16FD
0000  0674  1205          .WORD   0                               ; 17FD
0000  0676  1206          .WORD   0                               ; 18FD
0000  0678  1207          .WORD   0                               ; 19FD
0000  067A  1208          .WORD   0                               ; 1AFD
0000  067C  1209          .WORD   0                               ; 1BFD
0000  067E  1210          .WORD   0                               ; 1CFD
0000  0680  1211          .WORD   0                               ; 1DFD
0000  0682  1212          .WORD   0                               ; 1EFD
0000  0684  1213          .WORD   0                               ; 1FFD
0000  0686  1214          .WORD   0                               ; 20FD
0000  0688  1215          .WORD   0                               ; 21FD
0000  068A  1216          .WORD   0                               ; 22FD
0000  068C  1217          .WORD   0                               ; 23FD
0000  068E  1218          .WORD   0                               ; 24FD
0000  0690  1219          .WORD   0                               ; 25FD
0000  0692  1220          .WORD   0                               ; 26FD
0000  0694  1221          .WORD   0                               ; 27FD
0000  0696  1222          .WORD   0                               ; 28FD
0000  0698  1223          .WORD   0                               ; 29FD
0000  069A  1224          .WORD   0                               ; 2AFD
0000  069C  1225          .WORD   0                               ; 2BFD
0000  069E  1226          .WORD   0                               ; 2CFD
0000  06A0  1227          .WORD   0                               ; 2DFD
0000  06A2  1228          .WORD   0                               ; 2EFD
0000  06A4  1229          .WORD   0                               ; 2FFD
0000  06A6  1230          .WORD   0                               ; 30FD
0000  06A8  1231          .WORD   0                               ; 31FD
0388' 06AA  1232          .WORD   PATRN_CVTDH-TAB_2BYTE           ; 32FD  CVTDH
038B' 06AC  1233          .WORD   PATRN_CVTGF-TAB_2BYTE           ; 33FD  CVTGF
0000  06AE  1234          .WORD   0                               ; 34FD
0000  06B0  1235          .WORD   0                               ; 35FD
0000  06B2  1236          .WORD   0                               ; 36FD
0000  06B4  1237          .WORD   0                               ; 37FD
0000  06B6  1238          .WORD   0                               ; 38FD
0000  06B8  1239          .WORD   0                               ; 39FD
0000  06BA  1240          .WORD   0                               ; 3AFD
0000  06BC  1241          .WORD   0                               ; 3BFD
0000  06BE  1242          .WORD   0                               ; 3CFD
0000  06C0  1243          .WORD   0                               ; 3DFD
0000  06C2  1244          .WORD   0                               ; 3EFD
0000  06C4  1245          .WORD   0                               ; 3FFD
038E' 06C6  1246          .WORD   PATRN_ADDG2-TAB_2BYTE           ; 40FD  ADDG2
0391' 06C8  1247          .WORD   PATRN_ADDG3-TAB_2BYTE           ; 41FD  ADDG3
038E' 06CA  1248          .WORD   PATRN_SUBG2-TAB_2BYTE           ; 42FD  SUBG2
0391' 06CC  1249          .WORD   PATRN_SUBG3-TAB_2BYTE           ; 43FD  SUBG3
038E' 06CE  1250          .WORD   PATRN_MULG2-TAB_2BYTE           ; 44FD  MULG2
0391' 06D0  1251          .WORD   PATRN_MULG3-TAB_2BYTE           ; 45FD  MULG3
038E' 06D2  1252          .WORD   PATRN_DIVG2-TAB_2BYTE           ; 46FD  DIVG2
0391' 06D4  1253          .WORD   PATRN_DIVG3-TAB_2BYTE           ; 47FD  DIVG3
0395' 06D6  1254          .WORD   PATRN_CVTGB-TAB_2BYTE           ; 48FD  CVTGB
0398' 06D8  1255          .WORD   PATRN_CVTGW-TAB_2BYTE           ; 49FD  CVTGW
039B' 06DA  1256          .WORD   PATRN_CVTGL-TAB_2BYTE           ; 4AFD  CVTGL
039B' 06DC  1257          .WORD   PATRN_CVTRGL-TAB_2BYTE          ; 4BFD  CVTRGL
```

```
039E' 06DE  1258          .WORD   PATRN_CVTBG-TAB_2BYTE   ; 4CFD  CVTBG
03A1' 06E0  1259          .WORD   PATRN_CVTWG-TAB_2BYTE   ; 4DFD  CVTWG
03A4' 06E2  1260          .WORD   PATRN_CVTLG-TAB_2BYTE   ; 4EFD  CVTLG
03A7' 06E4  1261          .WORD   PATRN_ACBG-TAB_2BYTE    ; 4FFD  ACBG
03AC' 06E6  1262          .WORD   PATRN_MOVG-TAB_2BYTE    ; 50FD  MOVG
03AF' 06E8  1263          .WORD   PATRN_CMPG-TAB_2BYTE    ; 51FD  CMPG
03AC' 06EA  1264          .WORD   PATRN_MNEGG-TAB_2BYTE   ; 52FD  MNEGG
03B2' 06EC  1265          .WORD   PATRN_TSTG-TAB_2BYTE    ; 53FD  TSTG
03B4' 06EE  1266          .WORD   PATRN_EMODG-TAB_2BYTE   ; 54FD  EMODG
03BA' 06F0  1267          .WORD   PATRN_POLYG-TAB_2BYTE   ; 55FD  POLYG
03BE' 06F2  1268          .WORD   PATRN_CVTGH-TAB_2BYTE   ; 56FD  CVTGH
0000  06F4  1269          .WORD   0                       ; 57FD
0000  06F6  1270          .WORD   0                       ; 58FD
0000  06F8  1271          .WORD   0                       ; 59FD
0000  06FA  1272          .WORD   0                       ; 5AFD
0000  06FC  1273          .WORD   0                       ; 5BFD
0000  06FE  1274          .WORD   0                       ; 5CFD
0000  0700  1275          .WORD   0                       ; 5DFD
0000  0702  1276          .WORD   0                       ; 5EFD
0000  0704  1277          .WORD   0                       ; 5FFD
03C1' 0706  1278          .WORD   PATRN_ADDH2-TAB_2BYTE   ; 60FD  ADDH2
03C4' 0708  1279          .WORD   PATRN_ADDH3-TAB_2BYTE   ; 61FD  ADDH3
03C1' 070A  1280          .WORD   PATRN_SUBH2-TAB_2BYTE   ; 62FD  SUBH2
03C4' 070C  1281          .WORD   PATRN_SUBH3-TAB_2BYTE   ; 63FD  SUBH3
03C1' 070E  1282          .WORD   PATRN_MULH2-TAB_2BYTE   ; 64FD  MULH2
03C4' 0710  1283          .WORD   PATRN_MULH3-TAB_2BYTE   ; 65FD  MULH3
03C1' 0712  1284          .WORD   PATRN_DIVH2-TAB_2BYTE   ; 66FD  DIVH2
03C4' 0714  1285          .WORD   PATRN_DIVH3-TAB_2BYTE   ; 67FD  DIVH3
03C8' 0716  1286          .WORD   PATRN_CVTHB-TAB_2BYTE   ; 68FD  CVTHB
03CB' 0718  1287          .WORD   PATRN_CVTHW-TAB_2BYTE   ; 69FD  CVTHW
03CE' 071A  1288          .WORD   PATRN_CVTHL-TAB_2BYTE   ; 6AFD  CVTHL
03CE' 071C  1289          .WORD   PATRN_CVTRHL-TAB_2BYTE  ; 6BFD  CVTRHL
03D1' 071E  1290          .WORD   PATRN_CVTBH-TAB_2BYTE   ; 6CFD  CVTBH
03D4' 0720  1291          .WORD   PATRN_CVTWH-TAB_2BYTE   ; 6DFD  CVTWH
03D7' 0722  1292          .WORD   PATRN_CVTLH-TAB_2BYTE   ; 6EFD  CVTLH
03DA' 0724  1293          .WORD   PATRN_ACBH-TAB_2BYTE    ; 6FFD  ACBH
03DF' 0726  1294          .WORD   PATRN_MOVH-TAB_2BYTE    ; 70FD  MOVH
03E2' 0728  1295          .WORD   PATRN_CMPH-TAB_2BYTE    ; 71FD  CMPH
03DF' 072A  1296          .WORD   PATRN_MNEGH-TAB_2BYTE   ; 72FD  MNEGH
03E5' 072C  1297          .WORD   PATRN_TSTH-TAB_2BYTE    ; 73FD  TSTH
03E7' 072E  1298          .WORD   PATRN_EMODH-TAB_2BYTE   ; 74FD  EMODH
03ED' 0730  1299          .WORD   PATRN_POLYH-TAB_2BYTE   ; 75FD  POLYH
03F1' 0732  1300          .WORD   PATRN_CVTHG-TAB_2BYTE   ; 76FD  CVTHG
0000  0734  1301          .WORD   0                       ; 77FD
0000  0736  1302          .WORD   0                       ; 78FD
0000  0738  1303          .WORD   0                       ; 79FD
0000  073A  1304          .WORD   0                       ; 7AFD
0000  073C  1305          .WORD   0                       ; 7BFD
03F4' 073E  1306          .WORD   PATRN_CLRO-TAB_2BYTE    ; 7CFD  CLRO,CLRH
03F6' 0740  1307          .WORD   PATRN_MOVO-TAB_2BYTE    ; 7DFD  MOVO
03F9' 0742  1308          .WORD   PATRN_MOVAO-TAB_2BYTE   ; 7EFD  MOVAO,MOVAH
03FC' 0744  1309          .WORD   PATRN_PUSHAO-TAB_2BYTE  ; 7FFD  PUSHAO,PUSHAH
0000  0746  1310          .WORD   0                       ; 80FD
0000  0748  1311          .WORD   0                       ; 81FD
0000  074A  1312          .WORD   0                       ; 82FD
0000  074C  1313          .WORD   0                       ; 83FD
0000  074E  1314          .WORD   0                       ; 84FD
```

```
0000  0750  1315          .WORD   0                               ; 85FD
0000  0752  1316          .WORD   0                               ; 86FD
0000  0754  1317          .WORD   0                               ; 87FD
0000  0756  1318          .WORD   0                               ; 88FD
0000  0758  1319          .WORD   0                               ; 89FD
0000  075A  1320          .WORD   0                               ; 8AFD
0000  075C  1321          .WORD   0                               ; 8BFD
0000  075E  1322          .WORD   0                               ; 8CFD
0000  0760  1323          .WORD   0                               ; 8DFD
0000  0762  1324          .WORD   0                               ; 8EFD
0000  0764  1325          .WORD   0                               ; 8FFD
0000  0766  1326          .WORD   0                               ; 90FD
0000  0768  1327          .WORD   0                               ; 91FD
0000  076A  1328          .WORD   0                               ; 92FD
0000  076C  1329          .WORD   0                               ; 93FD
0000  076E  1330          .WORD   0                               ; 94FD
0000  0770  1331          .WORD   0                               ; 95FD
0000  0772  1332          .WORD   0                               ; 96FD
0000  0774  1333          .WORD   0                               ; 97FD
03FE' 0776  1334          .WORD   PATRN_CVTFH-TAB_2BYTE           ; 98FD  CVTFH
0401' 0778  1335          .WORD   PATRN_CVTFG-TAB_2BYTE           ; 99FD  CVTFG
0000  077A  1336          .WORD   0                               ; 9AFD
0000  077C  1337          .WORD   0                               ; 9BFD
0000  077E  1338          .WORD   0                               ; 9CFD
0000  0780  1339          .WORD   0                               ; 9DFD
0000  0782  1340          .WORD   0                               ; 9EFD
0000  0784  1341          .WORD   0                               ; 9FFD
0000  0786  1342          .WORD   0                               ; A0FD
0000  0788  1343          .WORD   0                               ; A1FD
0000  078A  1344          .WORD   0                               ; A2FD
0000  078C  1345          .WORD   0                               ; A3FD
0000  078E  1346          .WORD   0                               ; A4FD
0000  0790  1347          .WORD   0                               ; A5FD
0000  0792  1348          .WORD   0                               ; A6FD
0000  0794  1349          .WORD   0                               ; A7FD
0000  0796  1350          .WORD   0                               ; A8FD
0000  0798  1351          .WORD   0                               ; A9FD
0000  079A  1352          .WORD   0                               ; AAFD
0000  079C  1353          .WORD   0                               ; ABFD
0000  079E  1354          .WORD   0                               ; ACFD
0000  07A0  1355          .WORD   0                               ; ADFD
0000  07A2  1356          .WORD   0                               ; AEFD
0000  07A4  1357          .WORD   0                               ; AFFD
0000  07A6  1358          .WORD   0                               ; B0FD
0000  07A8  1359          .WORD   0                               ; B1FD
0000  07AA  1360          .WORD   0                               ; B2FD
0000  07AC  1361          .WORD   0                               ; B3FD
0000  07AE  1362          .WORD   0                               ; B4FD
0000  07B0  1363          .WORD   0                               ; B5FD
0000  07B2  1364          .WORD   0                               ; B6FD
0000  07B4  1365          .WORD   0                               ; B7FD
0000  07B6  1366          .WORD   0                               ; B8FD
0000  07B8  1367          .WORD   0                               ; B9FD
0000  07BA  1368          .WORD   0                               ; BAFD
0000  07BC  1369          .WORD   0                               ; BBFD
0000  07BE  1370          .WORD   0                               ; BCFD
0000  07C0  1371          .WORD   0                               ; BDFD
```

```
0000  07C2  1372          .WORD   0                       ; BEFD
0000  07C4  1373          .WORD   0                       ; BFFD
0000  07C6  1374          .WORD   0                       ; C0FD
0000  07C8  1375          .WORD   0                       ; C1FD
0000  07CA  1376          .WORD   0                       ; C2FD
0000  07CC  1377          .WORD   0                       ; C3FD
0000  07CE  1378          .WORD   0                       ; C4FD
0000  07D0  1379          .WORD   0                       ; C5FD
0000  07D2  1380          .WORD   0                       ; C6FD
0000  07D4  1381          .WORD   0                       ; C7FD
0000  07D6  1382          .WORD   0                       ; C8FD
0000  07D8  1383          .WORD   0                       ; C9FD
0000  07DA  1384          .WORD   0                       ; CAFD
0000  07DC  1385          .WORD   0                       ; CBFD
0000  07DE  1386          .WORD   0                       ; CCFD
0000  07E0  1387          .WORD   0                       ; CDFD
0000  07E2  1388          .WORD   0                       ; CEFD
0000  07E4  1389          .WORD   0                       ; CFFD
0000  07E6  1390          .WORD   0                       ; D0FD
0000  07E8  1391          .WORD   0                       ; D1FD
0000  07EA  1392          .WORD   0                       ; D2FD
0000  07EC  1393          .WORD   0                       ; D3FD
0000  07EE  1394          .WORD   0                       ; D4FD
0000  07F0  1395          .WORD   0                       ; D5FD
0000  07F2  1396          .WORD   0                       ; D6FD
0000  07F4  1397          .WORD   0                       ; D7FD
0000  07F6  1398          .WORD   0                       ; D8FD
0000  07F8  1399          .WORD   0                       ; D9FD
0000  07FA  1400          .WORD   0                       ; DAFD
0000  07FC  1401          .WORD   0                       ; DBFD
0000  07FE  1402          .WORD   0                       ; DCFD
0000  0800  1403          .WORD   0                       ; DDFD
0000  0802  1404          .WORD   0                       ; DEFD
0000  0804  1405          .WORD   0                       ; DFFD
0000  0806  1406          .WORD   0                       ; E0FD
0000  0808  1407          .WORD   0                       ; E1FD
0000  080A  1408          .WORD   0                       ; E2FD
0000  080C  1409          .WORD   0                       ; E3FD
0000  080E  1410          .WORD   0                       ; E4FD
0000  0810  1411          .WORD   0                       ; E5FD
0000  0812  1412          .WORD   0                       ; E6FD
0000  0814  1413          .WORD   0                       ; E7FD
0000  0816  1414          .WORD   0                       ; E8FD
0000  0818  1415          .WORD   0                       ; E9FD
0000  081A  1416          .WORD   0                       ; EAFD
0000  081C  1417          .WORD   0                       ; EBFD
0000  081E  1418          .WORD   0                       ; ECFD
0000  0820  1419          .WORD   0                       ; EDFD
0000  0822  1420          .WORD   0                       ; EEFD
0000  0824  1421          .WORD   0                       ; EFFD
0000  0826  1422          .WORD   0                       ; F0FD
0000  0828  1423          .WORD   0                       ; F1FD
0000  082A  1424          .WORD   0                       ; F2FD
0000  082C  1425          .WORD   0                       ; F3FD
0000  082E  1426          .WORD   0                       ; F4FD
0000  0830  1427          .WORD   0                       ; F5FD
0404' 0832  1428          .WORD   PATRN_CVTHF-TAB_2BYTE   ; F6FD   CVTHF
```

```
0407'  0834  1429          .WORD   PATRN_CVTHD-TAB_2BYTE   ; F7FD  CVTHD
0000   0836  1430          .WORD   0                       ; F8FD
0000   0838  1431          .WORD   0                       ; F9FD
0000   083A  1432          .WORD   0                       ; FAFD
0000   083C  1433          .WORD   0                       ; FBFD
0000   083E  1434          .WORD   0                       ; FCFD
0000   0840  1435          .WORD   0                       ; FDFD
0000   0842  1436          .WORD   0                       ; FEFD
0000   0844  1437          .WORD   0                       ; FFFD
       0846  1438
       0846  1439          ;
```

```
                     1441 ;+
                     1442 ; Instruction operand patterns
                     1443 ;
                     1444 ; Each pattern is defined using the macro OPDEF whose arguments
                     1445 ; describe the access type and data type of the operands.  The operand
                     1446 ; codes are of the form "xy", where "x" is the access type and "y" is
                     1447 ; the data type.
                     1448 ;-
                     1449
0846                 1450 PATRN_HALT:
0846                 1451 PATRN_NOP:
0846                 1452 PATRN_REI:
0846                 1453 PATRN_BPT:
0846                 1454 PATRN_RET:
0846                 1455 PATRN_RSB:
0846                 1456 PATRN_LDPCTX:
0846                 1457 PATRN_SVPCTX:
0846                 1458 PATRN_XFC:
0846                 1459         OPDEF
0847                 1460
0847                 1461 PATRN_CVTPS:
0847                 1462 PATRN_CVTSP:
0847                 1463         OPDEF   RW,AB,RW,AB
084C                 1464
084C                 1465 PATRN_INDEX:
084C                 1466         OPDEF   RL,RL,RL,RL,RL,WL
0853                 1467
0853                 1468 PATRN_CRC:
0853                 1469         OPDEF   AB,RL,RW,AB
0858                 1470
0858                 1471 PATRN_PROBER:
0858                 1472 PATRN_PROBEW:
0858                 1473         OPDEF   RB,RW,AB
085C                 1474
085C                 1475 PATRN_INSQUE:
085C                 1476         OPDEF   AB,AB
085F                 1477
085F                 1478 PATRN_REMQUE:
085F                 1479         OPDEF   AB,WL
0862                 1480
0862                 1481 PATRN_BSBB:
0862                 1482 PATRN_BRB:
0862                 1483 PATRN_BNEQ:
0862                 1484 PATRN_BEQL:
0862                 1485 PATRN_BGTR:
0862                 1486 PATRN_BLEQ:
0862                 1487 PATRN_BGEQ:
0862                 1488 PATRN_BLSS:
0862                 1489 PATRN_BGTRU:
0862                 1490 PATRN_BLEQU:
0862                 1491 PATRN_BVC:
0862                 1492 PATRN_BVS:
0862                 1493 PATRN_BGEQU:
0862                 1494 PATRN_BLSSU:
0862                 1495         OPDEF   BB
0864                 1496
0864                 1497 PATRN_JSB:
```

```
0864  1498 PATRN_JMP:
0864  1499 PATRN_PUSHAB:
0864  1500         OPDEF   AB
0866  1501
0866  1502 PATRN_ADDP4:
0866  1503 PATRN_SUBP4:
0866  1504 PATRN_CMPP4:
0866  1505 PATRN_MATCHC:
0866  1506         OPDEF   RW,AB,RW,AB
086B  1507
086B  1508 PATRN_ADDP6:
086B  1509 PATRN_SUBP6:
086B  1510 PATRN_MULP:
086B  1511 PATRN_DIVP:
086B  1512         OPDEF   RW,AB,RW,AB,RW,AB
0872  1513
0872  1514 PATRN_CVTPT:
0872  1515 PATRN_CVTTP:
0872  1516         OPDEF   RW,AB,AB,RW,AB
0878  1517
0878  1518 PATRN_MOVC3:
0878  1519 PATRN_CMPC3:
0878  1520         OPDEF   RW,AB,AB
087C  1521
087C  1522 PATRN_SCANC:
087C  1523 PATRN_SPANC:
087C  1524         OPDEF   RW,AB,AB,RB
0881  1525
0881  1526 PATRN_MOVC5:
0881  1527 PATRN_CMPC5:
0881  1528         OPDEF   RW,AB,RB,RW,AB
0887  1529
0887  1530 PATRN_MOVTC:
0887  1531 PATRN_MOVTUC:
0887  1532         OPDEF   RW,AB,RB,AB,RW,AB
088E  1533
088E  1534 PATRN_BSBW:
088E  1535 PATRN_BRW:
088E  1536         OPDEF   BW
0890  1537
0890  1538 PATRN_CVTWL:
0890  1539 PATRN_MOVZWL:
0890  1540         OPDEF   RW,WL
0893  1541
0893  1542 PATRN_CVTWB:
0893  1543         OPDEF   RW,WB
0896  1544
0896  1545 PATRN_MOVP:
0896  1546 PATRN_CMPP3:
0896  1547         OPDEF   RW,AB,AB
089A  1548
089A  1549 PATRN_CVTPL:
089A  1550         OPDEF   RW,AB,WL
089E  1551
089E  1552 PATRN_EDITPC:
089E  1553         OPDEF   RW,AB,AB,AB
08A3  1554
```

```
08A3  1555 PATRN_LOCC:
08A3  1556 PATRN_SKPC:
08A3  1557          OPDEF   RB,RW,AB
08A7  1558
08A7  1559 PATRN_ACBW:
08A7  1560          OPDEF   RW,RW,MW,BW
08AC  1561
08AC  1562 PATRN_MOVAW:
08AC  1563          OPDEF   AW,WL
08AF  1564
08AF  1565 PATRN_PUSHAW:
08AF  1566          OPDEF   AW
08B1  1567
08B1  1568 PATRN_ADDF2:
08B1  1569 PATRN_SUBF2:
08B1  1570 PATRN_MULF2:
08B1  1571 PATRN_DIVF2:
08B1  1572          OPDEF   RF,MF
08B4  1573
08B4  1574 PATRN_ADDF3:
08B4  1575 PATRN_SUBF3:
08B4  1576 PATRN_MULF3:
08B4  1577 PATRN_DIVF3:
08B4  1578          OPDEF   RF,RF,WF
08B8  1579
08B8  1580 PATRN_CVTFB:
08B8  1581          OPDEF   RF,WB
08BB  1582
08BB  1583 PATRN_CVTFW:
08BB  1584          OPDEF   RF,WW
08BE  1585
08BE  1586 PATRN_CVTFL:
08BE  1587 PATRN_CVTRFL:
08BE  1588          OPDEF   RF,WL
08C1  1589
08C1  1590 PATRN_CVTBF:
08C1  1591          OPDEF   RB,WF
08C4  1592
08C4  1593 PATRN_CVTWF:
08C4  1594          OPDEF   RW,WF
08C7  1595
08C7  1596 PATRN_CVTLF:
08C7  1597          OPDEF   RL,WF
08CA  1598
08CA  1599 PATRN_ACBF:
08CA  1600          OPDEF   RF,RF,MF,BW
08CF  1601
08CF  1602 PATRN_MOVF:
08CF  1603 PATRN_MNEGF:
08CF  1604          OPDEF   RF,WF
08D2  1605
08D2  1606 PATRN_CMPF:
08D2  1607          OPDEF   RF,RF
08D5  1608
08D5  1609 PATRN_TSTF:
08D5  1610          OPDEF   RF
08D7  1611
```

```
08D7  1612 PATRN_EMODF:
08D7  1613          OPDEF   RF,RB,RF,WL,WF
08DD  1614
08DD  1615 PATRN_POLYF:
08DD  1616          OPDEF   RF,RW,AB
08E1  1617
08E1  1618 PATRN_CVTFD:
08E1  1619          OPDEF   RF,WD
08E4  1620
08E4  1621 PATRN_ADAWI:
08E4  1622 PATRN_ADDW2:
08E4  1623 PATRN_SUBW2:
08E4  1624 PATRN_MULW2:
08E4  1625 PATRN_DIVW2:
08E4  1626 PATRN_BISW2:
08E4  1627 PATRN_BICW2:
08E4  1628 PATRN_XORW2:
08E4  1629          OPDEF   RW,MW
08E7  1630
08E7  1631 PATRN_INSQHI:
08E7  1632 PATRN_INSQTI:
08E7  1633          OPDEF   AB,AQ
08EA  1634
08EA  1635 PATRN_REMQHI:
08EA  1636 PATRN_REMQTI:
08EA  1637          OPDEF   AQ,WL
08ED  1638
08ED  1639 PATRN_ADDD2:
08ED  1640 PATRN_SUBD2:
08ED  1641 PATRN_MULD2:
08ED  1642 PATRN_DIVD2:
08ED  1643          OPDEF   RD,MD
08F0  1644
08F0  1645 PATRN_ADDD3:
08F0  1646 PATRN_SUBD3:
08F0  1647 PATRN_MULD3:
08F0  1648 PATRN_DIVD3:
08F0  1649          OPDEF   RD,RD,WD
08F4  1650
08F4  1651 PATRN_CVTDB:
08F4  1652          OPDEF   RD,WB
08F7  1653
08F7  1654 PATRN_CVTDW:
08F7  1655          OPDEF   RD,WW
08FA  1656
08FA  1657 PATRN_CVTDL:
08FA  1658 PATRN_CVTRDL:
08FA  1659          OPDEF   RD,WL
08FD  1660
08FD  1661 PATRN_CVTBD:
08FD  1662          OPDEF   RB,WD
0900  1663
0900  1664 PATRN_CVTWD:
0900  1665          OPDEF   RW,WD
0903  1666
0903  1667 PATRN_CVTLD:
0903  1668          OPDEF   RL,WD
```

```
0906  1669
0906  1670 PATRN_ACBD:
0906  1671          OPDEF   RD,RD,MD,BW
090B  1672
090B  1673 PATRN_MOVD:
090B  1674 PATRN_MNEGD:
090B  1675          OPDEF   RD,WD
090E  1676
090E  1677 PATRN_CMPD:
090E  1678          OPDEF   RD,RD
0911  1679
0911  1680 PATRN_TSTD:
0911  1681          OPDEF   RD
0913  1682
0913  1683 PATRN_EMODD:
0913  1684          OPDEF   RD,RB,RD,WL,WD
0919  1685
0919  1686 PATRN_POLYD:
0919  1687          OPDEF   RD,RW,AB
091D  1688
091D  1689 PATRN_CVTDF:
091D  1690          OPDEF   RD,WF
0920  1691
0920  1692 PATRN_ASHL:
0920  1693          OPDEF   RB,RL,WL
0924  1694
0924  1695 PATRN_ASHQ:
0924  1696          OPDEF   RB,RQ,WQ
0928  1697
0928  1698 PATRN_EMUL:
0928  1699          OPDEF   RL,RL,RL,WQ
092D  1700
092D  1701 PATRN_EDIV:
092D  1702          OPDEF   RL,RQ,WL,WL
0932  1703
0932  1704 PATRN_CLRQ:
0932  1705          OPDEF   WQ
0934  1706
0934  1707 PATRN_MOVQ:
0934  1708          OPDEF   RQ,WQ
0937  1709
0937  1710 PATRN_MOVAQ:
0937  1711          OPDEF   AQ,WL
093A  1712
093A  1713 PATRN_PUSHAQ:
093A  1714          OPDEF   AQ
093C  1715
093C  1716 PATRN_ADDB2:
093C  1717 PATRN_SUBB2:
093C  1718 PATRN_MULB2:
093C  1719 PATRN_DIVB2:
093C  1720 PATRN_BISB2:
093C  1721 PATRN_BICB2:
093C  1722 PATRN_XORB2:
093C  1723          OPDEF   RB,MB
093F  1724
093F  1725 PATRN_ADDB3:
```

```
093F  1726 PATRN_SUBB3:
093F  1727 PATRN_MULB3:
093F  1728 PATRN_DIVB3:
093F  1729 PATRN_BISB3:
093F  1730 PATRN_BICB3:
093F  1731 PATRN_XORB3:
093F  1732          OPDEF   RB,RB,WB
0943  1733
0943  1734 PATRN_MNEGB:
0943  1735 PATRN_MOVB:
0943  1736 PATRN_MCOMB:
0943  1737          OPDEF   RB,WB
0946  1738
0946  1739 PATRN_CASEB:
0946  1740          OPDEF   RB,RB,RB        ; User must understand branch table
094A  1741
094A  1742 PATRN_CMPB:
094A  1743 PATRN_BITB:
094A  1744          OPDEF   RB,RB
094D  1745
094D  1746 PATRN_CLRB:
094D  1747          OPDEF   WB
094F  1748
094F  1749 PATRN_TSTB:
094F  1750          OPDEF   RB
0951  1751
0951  1752 PATRN_INCB:
0951  1753 PATRN_DECB:
0951  1754          OPDEF   MB
0953  1755
0953  1756 PATRN_CVTBL:
0953  1757 PATRN_MOVZBL:
0953  1758          OPDEF   RB,WL
0956  1759
0956  1760 PATRN_CVTBW:
0956  1761 PATRN_MOVZBW:
0956  1762          OPDEF   RB,WW
0959  1763
0959  1764 PATRN_ROTL:
0959  1765          OPDEF   RB,RL,WL
095D  1766
095D  1767 PATRN_ACBB:
095D  1768          OPDEF   RB,RB,MB,BW
0962  1769
0962  1770 PATRN_MOVAB:
0962  1771          OPDEF   AB,WL
0965  1772
0965  1773 PATRN_ADDW3:
0965  1774 PATRN_SUBW3:
0965  1775 PATRN_MULW3:
0965  1776 PATRN_DIVW3:
0965  1777 PATRN_BISW3:
0965  1778 PATRN_BICW3:
0965  1779 PATRN_XORW3:
0965  1780          OPDEF   RW,RW,WW
0969  1781
0969  1782 PATRN_MNEGW:
```

```
0969  1783 PATRN_MOVW:
0969  1784 PATRN_MCOMW:
0969  1785          OPDEF   RW,WW
096C  1786
096C  1787 PATRN_CASEW:
096C  1788          OPDEF   RW,RW,RW        ; User must understand branch table
0970  1789
0970  1790 PATRN_CMPW:
0970  1791 PATRN_BITW:
0970  1792          OPDEF   RW,RW
0973  1793
0973  1794 PATRN_CLRW:
0973  1795          OPDEF   WW
0975  1796
0975  1797 PATRN_TSTW:
0975  1798 PATRN_BISPSW:
0975  1799 PATRN_BICPSW:
0975  1800 PATRN_POPR:
0975  1801 PATRN_PUSHR:
0975  1802 PATRN_CHMK:
0975  1803 PATRN_CHME:
0975  1804 PATRN_CHMS:
0975  1805 PATRN_CHMU:
0975  1806          OPDEF   RW
0977  1807
0977  1808 PATRN_INCW:
0977  1809 PATRN_DECW:
0977  1810          OPDEF   MW
0979  1811
0979  1812 PATRN_ADDL2:
0979  1813 PATRN_SUBL2:
0979  1814 PATRN_MULL2:
0979  1815 PATRN_DIVL2:
0979  1816 PATRN_BISL2:
0979  1817 PATRN_BICL2:
0979  1818 PATRN_XORL2:
0979  1819 PATRN_ADWC:
0979  1820 PATRN_SBWC:
0979  1821          OPDEF   RL,ML
097C  1822
097C  1823 PATRN_ADDL3:
097C  1824 PATRN_SUBL3:
097C  1825 PATRN_MULL3:
097C  1826 PATRN_DIVL3:
097C  1827 PATRN_BISL3:
097C  1828 PATRN_BICL3:
097C  1829 PATRN_XORL3:
097C  1830          OPDEF   RL,RL,WL
0980  1831
0980  1832 PATRN_MNEGL:
0980  1833 PATRN_MOVL:
0980  1834 PATRN_MCOML:
0980  1835 PATRN_MFPR:
0980  1836          OPDEF   RL,WL
0983  1837
0983  1838 PATRN_CASEL:
0983  1839          OPDEF   RL,RL,RL        ; User must understand branch table
```

```
                     1840
                     1841 PATRN_CMPL:
                     1842 PATRN_BITL:
                     1843 PATRN_MTPR:
0987                 1844         OPDEF   RL,RL
098A                 1845
098A                 1846 PATRN_CLRL:
098A                 1847 PATRN_MOVPSL:
098A                 1848         OPDEF   WL
098C                 1849
098C                 1850 PATRN_TSTL:
098C                 1851 PATRN_PUSHL:
098C                 1852         OPDEF   RL
098E                 1853
098E                 1854 PATRN_INCL:
098E                 1855 PATRN_DECL:
098E                 1856         OPDEF   ML
0990                 1857
0990                 1858 PATRN_MOVAL:
0990                 1859         OPDEF   AL,WL
0993                 1860
0993                 1861 PATRN_PUSHAL:
0993                 1862         OPDEF   AL
0995                 1863
0995                 1864 PATRN_BBS:
0995                 1865 PATRN_BBC:
0995                 1866 PATRN_BBSS:
0995                 1867 PATRN_BBCS:
0995                 1868 PATRN_BBSC:
0995                 1869 PATRN_BBCC:
0995                 1870 PATRN_BBSSI:
0995                 1871 PATRN_BBCCI:
0995                 1872         OPDEF   RL,VB,BB
0999                 1873
0999                 1874 PATRN_BLBS:
0999                 1875 PATRN_BLBC:
0999                 1876         OPDEF   RL,BB
099C                 1877
099C                 1878 PATRN_FFS:
099C                 1879 PATRN_FFC:
099C                 1880 PATRN_EXTV:
099C                 1881 PATRN_EXTZV:
099C                 1882         OPDEF   RL,RB,VB,WL
09A1                 1883
09A1                 1884 PATRN_CMPV:
09A1                 1885 PATRN_CMPZV:
09A1                 1886         OPDEF   RL,RB,VB,RL
09A6                 1887
09A6                 1888 PATRN_INSV:
09A6                 1889         OPDEF   RL,RL,RB,VB
09AB                 1890
09AB                 1891 PATRN_ACBL:
09AB                 1892         OPDEF   RL,RL,ML,BW
09B0                 1893
09B0                 1894 PATRN_AOBLSS:
09B0                 1895 PATRN_AOBLEQ:
09B0                 1896         OPDEF   RL,ML,BB
```

```
09B4  1897
09B4  1898 PATRN_SOBGEQ:
09B4  1899 PATRN_SOBGTR:
09B4  1900          OPDEF   ML,BB
09B7  1901
09B7  1902 PATRN_CVTLB:
09B7  1903          OPDEF   RL,WB
09BA  1904
09BA  1905 PATRN_CVTLW:
09BA  1906          OPDEF   RL,WW
09BD  1907
09BD  1908 PATRN_ASHP:
09BD  1909          OPDEF   RB,RW,AB,RB,RW,AB
09C4  1910
09C4  1911 PATRN_CVTLP:
09C4  1912          OPDEF   RL,RW,AB
09C8  1913
09C8  1914 PATRN_CALLS:
09C8  1915          OPDEF   RL,AB
09CB  1916
09CB  1917 PATRN_CALLG:
09CB  1918          OPDEF   AB,AB
09CE  1919
09CE  1920 PATRN_CVTDH:
09CE  1921          OPDEF   RD,WH
09D1  1922
09D1  1923 PATRN_CVTGF:
09D1  1924          OPDEF   RG,WF
09D4  1925
09D4  1926 PATRN_ADDG2:
09D4  1927 PATRN_SUBG2:
09D4  1928 PATRN_MULG2:
09D4  1929 PATRN_DIVG2:
09D4  1930          OPDEF   RG,MG
09D7  1931
09D7  1932 PATRN_ADDG3:
09D7  1933 PATRN_SUBG3:
09D7  1934 PATRN_MULG3:
09D7  1935 PATRN_DIVG3:
09D7  1936          OPDEF   RG,RG,WG
09DB  1937
09DB  1938 PATRN_CVTGB:
09DB  1939          OPDEF   RG,WB
09DE  1940
09DE  1941 PATRN_CVTGW:
09DE  1942          OPDEF   RG,WW
09E1  1943
09E1  1944 PATRN_CVTGL:
09E1  1945 PATRN_CVTRGL:
09E1  1946          OPDEF   RG,WL
09E4  1947
09E4  1948 PATRN_CVTBG:
09E4  1949          OPDEF   RB,WG
09E7  1950
09E7  1951 PATRN_CVTWG:
09E7  1952          OPDEF   RW,WG
09EA  1953
```

```
                        1954 PATRN_CVTLG:
                        1955         OPDEF   RL,WG
                        1956
                        1957 PATRN_ACBG:
                        1958         OPDEF   RG,RG,MG,BW
                        1959
                        1960 PATRN_MOVG:
                        1961 PATRN_MNEGG:
                        1962         OPDEF   RG,WG
                        1963
                        1964 PATRN_CMPG:
                        1965         OPDEF   RG,RG
                        1966
                        1967 PATRN_TSTG:
                        1968         OPDEF   RG
                        1969
                        1970 PATRN_EMODG:
                        1971         OPDEF   RG,RW,RG,WL,WG
                        1972
                        1973 PATRN_POLYG:
                        1974         OPDEF   RG,RW,AB
                        1975
                        1976 PATRN_CVTGH:
                        1977         OPDEF   RG,WH
                        1978
                        1979 PATRN_ADDH2:
                        1980 PATRN_SUBH2:
                        1981 PATRN_MULH2:
                        1982 PATRN_DIVH2:
                        1983         OPDEF   RH,MH
                        1984
                        1985 PATRN_ADDH3:
                        1986 PATRN_SUBH3:
                        1987 PATRN_MULH3:
                        1988 PATRN_DIVH3:
                        1989         OPDEF   RH,RH,WH
                        1990
                        1991 PATRN_CVTHB:
                        1992         OPDEF   RH,WB
                        1993
                        1994 PATRN_CVTHW:
                        1995         OPDEF   RH,WW
                        1996
                        1997 PATRN_CVTHL:
                        1998 PATRN_CVTRHL:
                        1999         OPDEF   RH,WL
                        2000
                        2001 PATRN_CVTBH:
                        2002         OPDEF   RB,WH
                        2003
                        2004 PATRN_CVTWH:
                        2005         OPDEF   RW,WH
                        2006
                        2007 PATRN_CVTLH:
                        2008         OPDEF   RL,WH
                        2009
                        2010 PATRN_ACBH:
```

```
                        2011          OPDEF   RH,RH,MH,BW
                        2012
                        2013 PATRN_MOVH:
                        2014 PATRN_MNEGH:
                        2015          OPDEF   RH,WH
                        2016
                        2017 PATRN_CMPH:
                        2018          OPDEF   RH,RH
                        2019
                        2020 PATRN_TSTH:
                        2021          OPDEF   RH
                        2022
                        2023 PATRN_EMODH:
                        2024          OPDEF   RH,RW,RH,WL,WH
                        2025
                        2026 PATRN_POLYH:
                        2027          OPDEF   RH,RW,AB
                        2028
                        2029 PATRN_CVTHG:
                        2030          OPDEF   RH,WG
                        2031
                        2032 PATRN_CLRO:
                        2033          OPDEF   WO
                        2034
                        2035 PATRN_MOVO:
                        2036          OPDEF   RO,WO
                        2037
                        2038 PATRN_MOVAO:
                        2039          OPDEF   AO,WL
                        2040
                        2041 PATRN_PUSHAO:
                        2042          OPDEF   AO
                        2043
                        2044 PATRN_CVTFH:
                        2045          OPDEF   RF,WH
                        2046
                        2047 PATRN_CVTFG:
                        2048          OPDEF   RF,WG
                        2049
                        2050 PATRN_CVTHF:
                        2051          OPDEF   RH,WF
                        2052
                        2053 PATRN_CVTHD:
                        2054          OPDEF   RH,WD
```

```
0A50  2056          .SBTTL Operand Decoding Routines
0A50  2057
0A50  2058          ;       ************************************************************************
0A50  2059          ;       *                                                                      *
0A50  2060          ;       *                                                                      *
0A50  2061          ;       *          Routines for Scanning Instruction Operands                  *
0A50  2062          ;       *                                                                      *
0A50  2063          ;       *                                                                      *
0A50  2064          ;       ************************************************************************
0A50  2065          ;
0A50  2066          ;
0A50  2067          ;       Introduction
0A50  2068          ;       ------------
0A50  2069          ;
0A50  2070          ;           The following section contains a set of routines for
0A50  2071          ;       scanning the operands of an instruction and determining the
0A50  2072          ;       values and locations of operands. The code contains full error
0A50  2073          ;       checking and also checks for the situations that the
0A50  2074          ;       architecture considers to be unpredictable.
0A50  2075          ;
0A50  2076          ;       Operand Scanning Routines
0A50  2077          ;       -------------------------
0A50  2078          ;
0A50  2079          ;       The operand scanning routines are entered by loading the
0A50  2080          ;       access type into R8, the data type into R9, and then
0A50  2081          ;       performing a JSB-type branch to GET_SPECIFIER.
0A50  2082          ;
0A50  2083          ;           When the routines are entered they scan the next
0A50  2084          ;       instruction operand starting at the value of the user's PC
0A50  2085          ;       and check the operand for validity. If any exceptions are
0A50  2086          ;       detected during operand scanning they are processed immediatly
0A50  2087          ;       and the routines do not return. Any changes that are made to
0A50  2088          ;       any of the registers (including PC) are recorded in the change
0A50  2089          ;       words so faults will be handled properly.
0A50  2090          ;
0A50  2091          ;           If the operand access type is READ, MODIFY or FIELD,
0A50  2092          ;       ADDRESS or BRANCH, the address of the value is placed in the
0A50  2093          ;       appropriate element of the READ_ADDRS array.  If the operand
0A50  2094          ;       is also immediate mode or a register, its value is copied to
0A50  2095          ;       the READ_OPERANDS array and READ_ADDRS points to that location.
0A50  2096          ;       This is to prevent later operand specifiers from modifying
0A50  2097          ;       registers previously used as operands, and gives a place to
0A50  2098          ;       store immediate mode operands.
0A50  2099          ;
0A50  2100          ;           If the operand access type is WRITE, MODIFY, ADDRESS, or
0A50  2101          ;       FIELD, the address of the value is placed in the appropriate
0A50  2102          ;       element of the WRITE_ADDRS array.  If the operand is a
0A50  2103          ;       register, a pointer to the appropriate emulated register
0A50  2104          ;       is used.  This permits correct implementation of the
0A50  2105          ;       "read all operands, then write all operands" rule of
0A50  2106          ;       the VAX architecture.
0A50  2107          ;
0A50  2108          ;       If the WRITE_ADDRS element is to be filled with a value that
0A50  2109          ;       addresses our local storage then it
0A50  2110          ;       is changed to an address that won't do any harm. This is
0A50  2111          ;       consistent with the notion that the area below the user's
0A50  2112          ;       stack pointer is being continually garbaged. This check is not
```

```
0A50  2113 ;
0A50  2114 ;
0A50  2115 ;
0A50  2116 ;
0A50  2117 ;
0A50  2118 ;
0A50  2119 ;
0A50  2120 ;
0A50  2121 ;
0A50  2122 ;
0A50  2123 ;
0A50  2124 ;
0A50  2125 ;
0A50  2126 ;
0A50  2127 ;
0A50  2128 ;
0A50  2129 ;
0A50  2130 ;
0A50  2131 ;
0A50  2132 ;
0A50  2133 ;
0A50  2134 ;
0A50  2135 ;
0A50  2136 ;
0A50  2137 ;
0A50  2138 ;
0A50  2139 ;
0A50  2140 ;
0A50  2141 ;
0A50  2142 ;
0A50  2143 ;
0A50  2144 ;
0A50  2145 ;
0A50  2146 ;
0A50  2147 ;
0A50  2148 ;
0A50  2149 ;
0A50  2150 ;
0A50  2151 ;
0A50  2152 ;
0A50  2153 ;
0A50  2154 ;
0A50  2155 ;
0A50  2156 ;
0A50  2157 ;
0A50  2158 ;
0A50  2159 ;
0A50  2160 ;
0A50  2161 ;
0A50  2162 ;
0A50  2163 ;
0A50  2164 ;
0A50  2165 ;
0A50  2166 ;
0A50  2167 ;
0A50  2168 ;
0A50  2169 ;
```

performed if flag bit V_REGISTER is set, which indicates that the operand is a register mode operand.

If the FPD bit is set in the PSL, the only effect of decoding an operand specifier is to move the PC.

Exceptions

The instruction operand scanning routines perform complete error checking and immediatly signal any exceptions detected. All of these exceptions are faults.

All fetches from memory done in scanning the instruction operand or in fetching the operand or operand address are probed and access violations are signaled if the probes fail. All of the addressing modes specified by the architecture to be reserved addressing modes or unpredictable are checked for and are signaled as reserved addressing modes if they are detected.

Routine Organization

GET_SPECIFIER loads the length of the data type into R10 and the operand specifier byte into R0. The high and low order nibbles of this byte are stored in R1 and R2. The register R7 which is reserved for the index modification is cleared. The routine now branches on the high order nibble to a routine which will handle the specific kind of operand.

For literals the values are expanded immediately.

For index mode operand specifiers, the index modification is computed and the next operand specifier byte is loaded into R0 and decomposed as before. Again we branch on the high order nibble but this time certain addressing modes which are illegal with indexing are checked for. Also for those addressing modes which change register values a check is made that the register is not the same as the index register.

For register mode operands the address of the emulated register is loaded into R11. A check is made that the operand does not contain PC. Then flag bit V_REGISTER is set and control passes to the operand reading routine.

For the remaining addressing modes the operand addresses are computed in a straightforward manner and loaded into R11. for some of these addressing modes the values of registers may be changed. These changes are reflected in the change words. When the operand address is computed control passes to the operand accessing routine.

For ADDRESS and FIELD access mode operands the operand accessing routine returns but for all others it probes the

```
                        ;
                        ;   operand address and also checks for writes into local
                        ;   storage unless V_REGISTER is set. If the operand is READ or
                        ;   MODIFY access control passes to the operand reading routine.
0A50  2170              ;
0A50  2171              ;
0A50  2172              ;
0A50  2173              ;
0A50  2174              ;
```

```
                                0A50  2176          ;
                                0A50  2177          ;       Table of Data Type Lengths
                                0A50  2178          ;
                                0A50  2179 LENGTHS:                                     ; table origin
                             01 0A50  2180          .BYTE   1                           ; 1 - byte
                             02 0A51  2181          .BYTE   2                           ; 2 - word
                             04 0A52  2182          .BYTE   4                           ; 3 - longword
                             08 0A53  2183          .BYTE   8                           ; 4 - quadword
                             10 0A54  2184          .BYTE   16                          ; 5 - octaword
                             04 0A55  2185          .BYTE   4                           ; 6 - F_floating
                             08 0A56  2186          .BYTE   8                           ; 7 - D_floating
                             08 0A57  2187          .BYTE   8                           ; 8 - G_floating
                             10 0A58  2188          .BYTE   16                          ; 9 - H_floating
                                0A59  2189          ;
                                0A59  2190          ;       Process the Next Operand Specifier
                                0A59  2191          ;
                                0A59  2192 GET_SPECIFIER:                               ; entrance
                      06   58   D1  0A59  2193          CMPL    R8,#LIB$K_DCFACC_B        ; Is this a branch address?
                           03   12  0A5C  2194          BNEQ    1$                        ; If not, skip
                         03F9   31  0A5E  2195          BRW     BRANCH_ACCESS             ; If so, go process it
                5A    EA AF49   9A  0A61  2196 1$:      MOVZBL  LENGTHS-1[R9],R10         ; R10 = data type length
                          57    D4  0A66  2197          CLRL    R7                        ; clear the index value
                   5B   50 AD   D0  0A68  2198          MOVL    REG_PC(FP),R11            ; R11 = specifier byte location
             6B    01   FD AD   0C  0A6C  2199          PROBER  MODE(FP),#1,(R11)         ; can we read the specifier byte ?
                          06   12  0A71  2200          BNEQ    2$                        ; yes - skip
                5A    01   D0  0A73  2201          MOVL    #1,R10                    ; R10 = size of probe
                        0472   30  0A76  2202          BSBW    READ_FAULT                ; process an access violation
                   50   50 BD   9A  0A79  2203 2$:      MOVZBL  @REG_PC(FP),R0            ; R0 = specifier byte
                        50 AD   D6  0A7D  2204          INCL    REG_PC(FP)                ; increment PC
          51    50    04   04    EF  0A80  2205          EXTZV   #4,#4,R0,R1               ; R1 = high order nibble of specifier
          52    50    04   00    EF  0A85  2206          EXTZV   #0,#4,R0,R2               ; R2 = low order nibble of specifier
               0F    00   51    CF  0A8A  2207          CASEL   R1,#0,#15                 ; branch on the high order nibble
                          0020' 0A8E  2208 3$:      .WORD   LITERAL_MODE-3$           ; 0 - literal mode
                          0020' 0A90  2209          .WORD   LITERAL_MODE-3$           ; 1 - literal mode
                          0020' 0A92  2210          .WORD   LITERAL_MODE-3$           ; 2 - literal mode
                          0020' 0A94  2211          .WORD   LITERAL_MODE-3$           ; 3 - literal mode
                          0084' 0A96  2212          .WORD   INDEX_MODE-3$             ; 4 - index mode
                          00F2' 0A98  2213          .WORD   REGISTER_MODE-3$          ; 5 - register mode
                          0177' 0A9A  2214          .WORD   REG_DEF_MODE-3$           ; 6 - register deferred mode
                          018E' 0A9C  2215          .WORD   DECR_MODE-3$              ; 7 - autodecrement mode
                          01AA' 0A9E  2216          .WORD   INCR_MODE-3$              ; 8 - autoincrement mode
                          01DB' 0AA0  2217          .WORD   INCR_DEF_MODE-3$          ; 9 - autoincrement deferred mode
                          0208' 0AA2  2218          .WORD   BYTE_DISP_MODE-3$         ; A - byte displacement mode
                          0230' 0AA4  2219          .WORD   BYTE_DEF_MODE-3$          ; B - byte displacement deferred mode
                          0266' 0AA6  2220          .WORD   WORD_DISP_MODE-3$         ; C - word displacement mode
                          028F' 0AA8  2221          .WORD   WORD_DEF_MODE-3$          ; D - word displacement deferred mode
                          02C6' 0AAA  2222          .WORD   LONG_DISP_MODE-3$         ; E - long displacement mode
                          02EF' 0AAC  2223          .WORD   LONG_DEF_MODE-3$          ; F - long displacement deferred mode
                                0AAE  2224          ;
                                0AAE  2225          ;       Process a Literal Mode Operand Specifier
                                0AAE  2226          ;
                                0AAE  2227 LITERAL_MODE:                                ; entrance
             43 54 AD    1B    E0  0AAE  2228          BBS     #PSL$V_FPD,PSL(FP),20$    ; Exit if FPD set
                5B   55    01   78  0AB3  2229          ASHL    #1,R5,R11                 ; Get address of location in
             5B    FEB8 CD4B   7E  0AB7  2230          MOVAQ   READ_OPERANDS(FP)[R11],R11 ; READ_OPERANDS to store literal
                04   01    58   CF  0ABD  2231          CASEL   R8,#1,#4                  ; branch on the access type
                          000B' 0AC1  2232 1$:      .WORD   2$-1$                     ; 1 - read only access
```

```
                        04F4' 0AC3  2233          .WORD   ADDRESS_FAULT-1$        ; 2 - modify access
                        04F4' 0AC5  2234          .WORD   ADDRESS_FAULT-1$        ; 3 - write only access
                        04F4' 0AC7  2235          .WORD   ADDRESS_FAULT-1$        ; 4 - address access
                        04F4' 0AC9  2236          .WORD   ADDRESS_FAULT-1$        ; 5 - field access
                          00  0ACB  2237          HALT                            ; 6 - branch access (should not occur)
            08   01   59   CF  0ACC  2238 2$:      CASEL   R9,#1,#8                ; branch on the data type
                        001B' 0AD0  2239 3$:      .WORD   6$-3$                   ; 1 - byte
                        001B' 0AD2  2240          .WORD   6$-3$                   ; 2 - word
                        001B' 0AD4  2241          .WORD   6$-3$                   ; 3 - longword
                        0015' 0AD6  2242          .WORD   5$-3$                   ; 4 - quadword
                        0012' 0AD8  2243          .WORD   4$-3$                   ; 5 - octaword
                        002A' 0ADA  2244          .WORD   8$-3$                   ; 6 - F_floating
                        0027' 0ADC  2245          .WORD   7$-3$                   ; 7 - D_floating
                        0032' 0ADE  2246          .WORD   9$-3$                   ; 8 - G_floating
                        003A' 0AE0  2247          .WORD   10$-3$                  ; 9 - H_floating
              08 AB   7C  0AE2  2248 4$:      CLRQ    8(R11)                  ; clear second quadword of value
              04 AB   D4  0AE5  2249 5$:      CLRL    4(R11)                  ; clear second longword of value
           6B   50   D0  0AE8  2250          MOVL    R0,(R11)                ; Move literal value
      FE78 CD45   5B   D0  0AEB  2251 6$:      MOVL    R11,READ_ADDRS(FP)[R5]  ; Set read operand address
           FE38 CD45   D4  0AF1  2252          CLRL    WRITE_ADDRS(FP)[R5]     ; Indicate no write operand address
                       05  0AF6  2253 20$:     RSB                             ; return
              04 AB   D4  0AF7  2254 7$:      CLRL    4(R11)                  ; clear second longword of value
        6B   50   04   78  0AFA  2255 8$:      ASHL    #4,R0,(R11)             ; position the literal bits
          E9 6B   0E   E3  0AFE  2256          BBCS    #14,(R11),6$            ; include exponent bias and return
        6B   50   01   78  0B02  2257 9$:      ASHL    #1,R0,(R11)             ; position the literal bits
          DB 6B   0E   E3  0B06  2258          BBCS    #14,(R11),5$            ; include exponent bias and finish up
        6B   50   1D   9C  0B0A  2259 10$:     ROTL    #29,R0,(R11)            ; position the literal bits
          D0 6B   0E   E3  0B0E  2260          BBCS    #14,(R11),4$            ; include exponent bias and finish up
                           0B12  2261          ;
                           0B12  2262          ;       Process an Index Mode Operand Specifier
                           0B12  2263          ;
                           0B12  2264 INDEX_MODE:                              ; entrance
                 52   0F   D1  0B12  2265          CMPL    #15,R2                  ; is the register PC ?
                      03   12  0B15  2266          BNEQ    1$                      ; no - skip
                    049B   31  0B17  2267          BRW     ADDRESS_FAULT           ; process the reserved addressing mode
          06 54 AD    1B   E0  0B1A  2268 1$:      BBS     #PSL$V_FPD,PSL(FP),11$  ; Skip if FPD set
        57   14 AD42    5A   C5  0B1F  2269          MULL3   R10,REG_R0(FP)[R2],R7   ; R7 = index address modification
                 53   52   D0  0B25  2270 11$:     MOVL    R2,R3                   ; save the register number
               5B   50 AD   D0  0B28  2271          MOVL    REG_PC(FP),R11          ; R11 = location of next byte
          6B   01   FD AD   0C  0B2C  2272          PROBER  MODE(FP),#1,(R11)       ; can we read the next byte ?
                      06   12  0B31  2273          BNEQ    2$                      ; yes - skip
                 5A   01   D0  0B33  2274          MOVL    #1,R10                  ; R10 = size of probe
                    03B2   30  0B36  2275          BSBW    READ_FAULT              ; process the access violation
               50   50 BD   9A  0B39  2276 2$:      MOVZBL  @REG_PC(FP),R0          ; R0 = next operand specifier
                 50 AD   D6  0B3D  2277          INCL    REG_PC(FP)              ; increment PC
        51   50   04   04   EF  0B40  2278          EXTZV   #4,#4,R0,R1             ; R1 = high order nibble of specifier
        52   50   04   00   EF  0B45  2279          EXTZV   #0,#4,R0,R2             ; R2 = low order nibble of specifier
             0F   00   51   CF  0B4A  2280          CASEL   R1,#0,#15               ; branch on the low order nibble
                        0467' 0B4E  2281 3$:      .WORD   ADDRESS_FAULT-3$        ; 0 - literal mode
                        0467' 0B50  2282          .WORD   ADDRESS_FAULT-3$        ; 1 - literal mode
                        0467' 0B52  2283          .WORD   ADDRESS_FAULT-3$        ; 2 - literal mode
                        0467' 0B54  2284          .WORD   ADDRESS_FAULT-3$        ; 3 - literal mode
                        0467' 0B56  2285          .WORD   ADDRESS_FAULT-3$        ; 4 - index mode
                        0467' 0B58  2286          .WORD   ADDRESS_FAULT-3$        ; 5 - register mode
                        0087' 0B5A  2287          .WORD   REG_DEF_MODE-3$         ; 6 - register deferred mode
                        0020' 0B5C  2288          .WORD   4$-3$                   ; 7 - autodecrement mode
                        0020' 0B5E  2289          .WORD   4$-3$                   ; 8 - autoincrement mode
```

```
                     0020' 0B60  2290          .WORD   4$-3$                   ; 9 - autoincrement deferred mode
                     0148' 0B62  2291          .WORD   BYTE_DISP_MODE-3$       ; A - byte displacement mode
                     0170' 0B64  2292          .WORD   BYTE_DEF_MODE-3$        ; B - byte displacement deferred mode
                     01A6' 0B66  2293          .WORD   WORD_DISP_MODE-3$       ; C - word displacement mode
                     01CF' 0B68  2294          .WORD   WORD_DEF_MODE-3$        ; D - word displacement deferred mode
                     0206' 0B6A  2295          .WORD   LONG_DISP_MODE-3$       ; E - long displacement mode
                     022F' 0B6C  2296          .WORD   LONG_DEF_MODE-3$        ; F - long displacement deferred mode
          53    52    D1  0B6E  2297 4$:       CMPL    R2,R3                   ; is register the same as index ?
                03    12  0B71  2298          BNEQ    5$                      ; no - skip
              043F    31  0B73  2299          BRW     ADDRESS_FAULT           ; process the reserved addressing mode
    02    07    51    CF  0B76  2300 5$:       CASEL   R1,#7,#2                ; branch on the high order nibble
                     00A2' 0B7A  2301 6$:       .WORD   DECR_MODE-6$            ; 7 - autodecrement mode
                     00BE' 0B7C  2302          .WORD   INCR_MODE-6$            ; 8 - autoincrement mode
                     00EF' 0B7E  2303          .WORD   INCR_DEF_MODE-6$        ; 9 - autoincrement deferred mode
                           0B80  2304          ;
                           0B80  2305          ;       Process a Register Mode Operand Specifier
                           0B80  2306          ;
                           0B80  2307 REGISTER_MODE:                           ; entrance
  2E    54    AD    1B    E0  0B80  2308          BBS     #PSL$V_FPD,PSL(FP),3$   ; Skip if FPD set
    FC    AD    01    88  0B85  2309          BISB2   #M_REGISTER,FLAGS(FP)   ; indicate a register mode operand
          53    6A42  DE  0B89  2310          MOVAL   (R10)[R2],R3            ; byte position following operand
          53    3C    D1  0B8D  2311          CMPL    #60,R3                  ; does the operand overlap PC ?
                03    18  0B90  2312          BGEQ    1$                      ; no - skip
              0420    31  0B92  2313          BRW     ADDRESS_FAULT           ; process the reserved addressing mode
    5B    14    AD42  DE  0B95  2314 1$:       MOVAL   REG_R0(FP)[R2],R11      ; R11 = location of user register
        FE78    CD45  D4  0B9A  2315          CLRL    READ_ADDRS(FP)[R5]      ; Initially no read operand
        FE38    CD45  D4  0B9F  2316          CLRL    WRITE_ADDRS(FP)[R5]     ; Initially no write operand
    04    01    58    CF  0BA4  2317          CASEL   R8,#1,#4                ; branch on the access type
                     001F' 0BA8  2318 2$:       .WORD   READ_REG-2$             ; 1 - read only access
                     0019' 0BAA  2319          .WORD   MODIFY_REG-2$           ; 2 - modify access
                     0012' 0BAC  2320          .WORD   WRITE_REG-2$            ; 3 - write access
                     040D' 0BAE  2321          .WORD   ADDRESS_FAULT-2$        ; 4 - address access
                     000C' 0BB0  2322          .WORD   FIELD_REG-2$            ; 5 - field access
                       00  0BB2  2323          HALT                            ; 6 - branch access (shouldn't occur)
                       05  0BB3  2324 3$:       RSB                             ; Return for FPD set
                           0BB4  2325          ;
                           0BB4  2326 FIELD_REG:
FE78    CD45    5B    D0  0BB4  2327          MOVL    R11,READ_ADDRS(FP)[R5]  ; Field reads/writes register
                           0BBA  2328 WRITE_REG:
FE38    CD45    5B    D0  0BBA  2329          MOVL    R11,WRITE_ADDRS(FP)[R5] ; Indicate write operand address
                       05  0BC0  2330          RSB
                           0BC1  2331 MODIFY_REG:
FE38    CD45    5B    D0  0BC1  2332          MOVL    R11,WRITE_ADDRS(FP)[R5] ; Indicate write operand address
                           0BC7  2333 READ_REG:
  50    55    01    78  0BC7  2334          ASHL    #1,R5,R0                ; Get address of location in
50    FEB8    CD40  7E  0BCB  2335          MOVAQ   READ_OPERANDS(FP)[R0],R0 ; READ_OPERANDS to store registers
FE78    CD45    50    D0  0BD1  2336          MOVL    R0, READ_ADDRS(FP)[R5]  ; Indicate write operand address
  08    01    59    CF  0BD7  2337          CASEL   R9,#1,#8                ; branch on the data type
                     0012' 0BDB  2338 1$:       .WORD   2$-1$                   ; 1 - byte
                     0016' 0BDD  2339          .WORD   3$-1$                   ; 2 - word
                     001A' 0BDF  2340          .WORD   4$-1$                   ; 3 - longword
                     0024' 0BE1  2341          .WORD   6$-1$                   ; 4 - quadword
                     001F' 0BE3  2342          .WORD   5$-1$                   ; 5 - octaword
                     001A' 0BE5  2343          .WORD   4$-1$                   ; 6 - F_floating
                     0024' 0BE7  2344          .WORD   6$-1$                   ; 7 - D_floating
                     0024' 0BE9  2345          .WORD   6$-1$                   ; 8 - G_floating
                     001F' 0BEB  2346          .WORD   5$-1$                   ; 9 - H_floating
```

```
                60   6B   98  0BED  2347 2$:      CVTBL   (R11),(R0)              ; save operand value
                     05   0BF0  2348              RSB                             ; return
                60   6B   32  0BF1  2349 3$:      CVTWL   (R11),(R0)              ; save operand value
                     05   0BF4  2350              RSB                             ; return
                60   6B   D0  0BF5  2351 4$:      MOVL    (R11),(R0)              ; save operand value
                     08   11  0BF8  2352              BRB     10$
          08 A0    08 AB   7D  0BFA  2353 5$:      MOVQ    8(R11),8(R0)            ; save high order quadword of value
                60   6B   7D  0BFF  2354 6$:      MOVQ    (R11),(R0)              ; save low order quadword of value
                   01E8   31  0C02  2355 10$:     BRW     ROPRAND_CHECK           ; Check for reserved operand
                          0C05  2356              ;
                          0C05  2357              ;       Process a Register Deferred Mode Operand Specifier
                          0C05  2358              ;
                          0C05  2359 REG_DEF_MODE:                                ; entrance
                52   0F   D1  0C05  2360              CMPL    #15,R2                  ; is the register PC ?
                     03   14  0C08  2361              BGTR    1$                      ; no - skip
                   03A8   31  0C0A  2362              BRW     ADDRESS_FAULT           ; process the reserved addressing mode
          09 54 AD   1B   E0  0C0D  2363 1$:      BBS     #PSL$V_FPD,PSL(FP),2$   ; Return if FPD set
       5B  14 AD42   57   C1  0C12  2364              ADDL3   R7,REG_R0(FP)[R2],R11   ; form the operand address
                   0199   31  0C18  2365              BRW     ACCESS_VALUE            ; finish establishing the access
                     05   0C1B  2366 2$:      RSB                             ; Return if FPD set
                          0C1C  2367              ;
                          0C1C  2368              ;       Process an Autodecrement Mode Operand Specifier
                          0C1C  2369              ;
                          0C1C  2370 DECR_MODE:                                   ; entrance
                52   0F   D1  0C1C  2371              CMPL    #15,R2                  ; is the register PC ?
                     03   14  0C1F  2372              BGTR    1$                      ; no - skip
                   0391   31  0C21  2373              BRW     ADDRESS_FAULT           ; process the reserved addressing mode
          0E 54 AD   1B   E0  0C24  2374 1$:      BBS     #PSL$V_FPD,PSL(FP),2$   ; Skip if FPD set
           14 AD42   5A   C2  0C29  2375              SUBL2   R10,REG_R0(FP)[R2]      ; subtract data size from register
       5B  14 AD42   57   C1  0C2E  2376              ADDL3   R7,REG_R0(FP)[R2],R11   ; form the operand address
                   017D   31  0C34  2377              BRW     ACCESS_VALUE            ; finish establishing the access
                     05   0C37  2378 2$:      RSB                             ; Return if FPD set
                          0C38  2379              ;
                          0C38  2380              ;       Process an Autoincrement Mode Operand Specifier
                          0C38  2381              ;
                          0C38  2382 INCR_MODE:                                   ; entrance
                52   0F   D1  0C38  2383              CMPL    #15,R2                  ; is the register PC ?
                     0F   14  0C3B  2384              BGTR    2$                      ; no - bypass
          04   01   58   CF  0C3D  2385              CASEL   R8,#1,#4                ; branch on the access type
                   000B' 0C41  2386 1$:      .WORD   2$-1$                   ; 1 - read only access
                   0374' 0C43  2387              .WORD   ADDRESS_FAULT-1$        ; 2 - modify access
                   0374' 0C45  2388              .WORD   ADDRESS_FAULT-1$        ; 3 - write only access
                   000B' 0C47  2389              .WORD   2$-1$                   ; 4 - address access
                   000B' 0C49  2390              .WORD   2$-1$                   ; 5 - field access
                     00  0C4B  2391              HALT                            ; 6 - branch access (shouldn't occur)
          0E 54 AD   1B   E0  0C4C  2392 2$:      BBS     #PSL$V_FPD,PSL(FP),3$   ; Skip if FPD set
       5B  14 AD42   57   C1  0C51  2393              ADDL3   R7,REG_R0(FP)[R2],R11   ; form the operand address
           14 AD42   5A   C0  0C57  2394              ADDL2   R10,REG_R0(FP)[R2]      ; add the data size to the register
                   0155   31  0C5C  2395              BRW     ACCESS_VALUE            ; finish establishing the access
                52   0F   D1  0C5F  2396 3$:      CMPL    R2, #15                 ; is the register PC?
                     04   12  0C62  2397              BNEQ    4$                      ; skip if not
             50 AD   5A   C0  0C64  2398              ADDL2   R10,REG_PC(FP)          ; Do the increment of PC anyway
                     05   0C68  2399 4$:      RSB                             ; Return if FPD set
                          0C69  2400              ;
                          0C69  2401              ;       Process an Autoincrement Deferred Mode Operand Specifier
                          0C69  2402              ;
                          0C69  2403 INCR_DEF_MODE:                               ; entrance
```

```
                1E 54 AD    1B   E0  0C69  2404          BBS     #PSL$V_FPD,PSL(FP),2$    ; skip if FPD set
                5B    14 AD42    D0  0C6E  2405          MOVL    REG_R0(FP)[R2],R11       ; R11 = register value
             6B    04    FD AD    0C  0C73  2406          PROBER  MODE(FP),#4,(R11)        ; can we read longword it addresses ?
                               06   12  0C78  2407          BNEQ    1$                       ; yes - skip
                         5A    04   D0  0C7A  2408          MOVL    #4,R10                   ; R10 = size of probe
                            026B   30  0C7D  2409          BSBW    READ_FAULT               ; process the access violation
                5B    6B    57   C1  0C80  2410 1$:       ADDL3   R7,(R11),R11             ; form the operand address
                14 AD42    04   C0  0C84  2411          ADDL2   #4,REG_R0(FP)[R2]        ; add longword size to the register
                            0128   31  0C89  2412          BRW     ACCESS_VALUE             ; finish establishing the access
                      0F    52   D1  0C8C  2413 2$:       CMPL    R2,#15                   ; is the register PC
                               04   12  0C8F  2414          BNEQ    3$                       ; skip if not
                   50 AD    04   C0  0C91  2415          ADDL2   #4,REG_PC(FP)            ; do autoincrement of PC anyway
                            05   0C95  2416 3$:       RSB                              ; return if FPD set
                                 0C96  2417          ;
                                 0C96  2418          ;       Process a Byte Displacement Mode Operand Specifier
                                 0C96  2419          ;
                                 0C96  2420 BYTE_DISP_MODE:                           ; entrance
                5B    50 AD    D0  0C96  2421          MOVL    REG_PC(FP),R11           ; R11 = location of displacement
             6B    01    FD AD    0C  0C9A  2422          PROBER  MODE(FP),#1,(R11)        ; can we read the displacement ?
                               06   12  0C9F  2423          BNEQ    1$                       ; yes - skip
                         5A    01   D0  0CA1  2424          MOVL    #1,R10                   ; R10 = size of probe
                            0244   30  0CA4  2425          BSBW    READ_FAULT               ; process the access violation
                   50 AD    D6   0CA7  2426 1$:       INCL    REG_PC(FP)               ; increment PC
                0E 54 AD    1B   E0  0CAA  2427          BBS     #PSL$V_FPD,PSL(FP),2$    ; skip if FPD set
                5B    6B    98   0CAF  2428          CVTBL   (R11),R11                ; R11 = displacement value
                5B    57    C0   0CB2  2429          ADDL2   R7,R11                   ; add the displacement to the index
                5B    14 AD42    C0   0CB5  2430          ADDL2   REG_R0(FP)[R2],R11       ; add the register to the result
                            00F7   31  0CBA  2431          BRW     ACCESS_VALUE             ; finish establishing the access
                            05   0CBD  2432 2$:       RSB                              ; Return if FPD set
                                 0CBE  2433          ;
                                 0CBE  2434          ;       Process a Byte Displacement Deferred Mode Operand Specifier
                                 0CBE  2435          ;
                                 0CBE  2436 BYTE_DEF_MODE:                            ; entrance
                5B    50 AD    D0  0CBE  2437          MOVL    REG_PC(FP),R11           ; R11 = location of displacement
             6B    01    FD AD    0C  0CC2  2438          PROBER  MODE(FP),#1,(R11)        ; can we read the displacement ?
                               06   12  0CC7  2439          BNEQ    1$                       ; yes - skip
                         5A    01   D0  0CC9  2440          MOVL    #1,R10                   ; R10 = size of probe
                            021C   30  0CCC  2441          BSBW    READ_FAULT               ; process the access violation
                   50 AD    D6   0CCF  2442 1$:       INCL    REG_PC(FP)               ; increment PC
                1C 54 AD    1B   E0  0CD2  2443          BBS     #PSL$V_FPD,PSL(FP),3$    ; skip if FPD set
                5B    6B    98   0CD7  2444          CVTBL   (R11),R11                ; R11 = displacement value
                5B    14 AD42    C0   0CDA  2445          ADDL2   REG_R0(FP)[R2],R11       ; add the register to the displacement
             6B    04    FD AD    0C  0CDF  2446          PROBER  MODE(FP),#4,(R11)        ; can we read longword it addresses ?
                               06   12  0CE4  2447          BNEQ    2$                       ; yes - skip
                         5A    04   D0  0CE6  2448          MOVL    #4,R10                   ; R10 = size of probe
                            01FF   30  0CE9  2449          BSBW    READ_FAULT               ; process the access fault
                5B    6B    57   C1  0CEC  2450 2$:       ADDL3   R7,(R11),R11             ; form the operand address
                            00C1   31  0CF0  2451          BRW     ACCESS_VALUE             ; finish establishing the access
                            05   0CF3  2452 3$:       RSB                              ; Return if FPD set
                                 0CF4  2453          ;
                                 0CF4  2454          ;       Process a Word Displacement Mode Operand Specifier
                                 0CF4  2455          ;
                                 0CF4  2456 WORD_DISP_MODE:                           ; entrance
                5B    50 AD    D0  0CF4  2457          MOVL    REG_PC(FP),R11           ; R11 = location of the displacement
             6B    02    FD AD    0C  0CF8  2458          PROBER  MODE(FP),#2,(R11)        ; can we read the displacement
                               06   12  0CFD  2459          BNEQ    1$                       ; yes - skip
                         5A    02   D0  0CFF  2460          MOVL    #2,R10                   ; R10 = size of probe
```

```
                01E6   30  0D02  2461          BSBW    READ_FAULT                      ; process the access violation
          50 AD   02   C0  0D05  2462 1$:     ADDL2   #2,REG_PC(FP)                   ; increment PC
       0E 54 AD   1B   E0  0D09  2463          BBS     #PSL$V_FPD,PSL(FP),2$           ; Skip if FPD set
             5B   6B   32  0D0E  2464          CVTWL   (R11),R11                       ; R11 = displacement value
             5B   57   C0  0D11  2465          ADDL2   R7,R11                          ; add the index to the displacement
          5B   14 AD42   C0  0D14  2466          ADDL2   REG_R0(FP)[R2],R11              ; add the register to the result
                0098   31  0D19  2467          BRW     ACCESS_VALUE                    ; finish establishing the access
                       05  0D1C  2468 2$:     RSB                                     ; Return if FPD set
                           0D1D  2469          ;
                           0D1D  2470          ;       Process a Word Displacement Deferred Mode Operand Specifier
                           0D1D  2471          ;
                           0D1D  2472 WORD_DEF_MODE:                                   ; entrance
          5B   50 AD   D0  0D1D  2473          MOVL    REG_PC(FP),R11                  ; R11 = location of the displacement
       6B 02   FD AD   0C  0D21  2474          PROBER  MODE(FP),#2,(R11)               ; can we read the displacement ?
                06   12  0D26  2475          BNEQ    1$                              ; yes - skip
             5A 02   D0  0D28  2476          MOVL    #2,R10                          ; R10 = size of probe
                01BD   30  0D2B  2477          BSBW    READ_FAULT                      ; process the access violation
          50 AD   02   C0  0D2E  2478 1$:     ADDL2   #2,REG_PC(FP)                   ; increment PC
       1C 54 AD   1B   E0  0D32  2479          BBS     #PSL$V_FPD,PSL(FP),3$           ; skip if FPD set
             5B   6B   32  0D37  2480          CVTWL   (R11),R11                       ; R11 = displacement value
          5B   14 AD42   C0  0D3A  2481          ADDL2   REG_R0(FP)[R2],R11              ; add the register to the displacement
       6B 04   FD AD   0C  0D3F  2482          PROBER  MODE(FP),#4,(R11)               ; can we read longword it addresses ?
                06   12  0D44  2483          BNEQ    2$                              ; yes - skip
             5A 04   D0  0D46  2484          MOVL    #4,R10                          ; R10 = size of probe
                019F   30  0D49  2485          BSBW    READ_FAULT                      ; process the access violation
          5B   6B 57   C1  0D4C  2486 2$:     ADDL3   R7,(R11),R11                    ; form the operand address
                0061   31  0D50  2487          BRW     ACCESS_VALUE                    ; finish establishing the access
                       05  0D53  2488 3$:     RSB                                     ; Return if FPD set
                           0D54  2489          ;
                           0D54  2490          ;       Process a Long Displacement Mode Operand Specifier
                           0D54  2491          ;
                           0D54  2492 LONG_DISP_MODE:                                  ; entrance
          5B   50 AD   D0  0D54  2493          MOVL    REG_PC(FP),R11                  ; R11 = location of the displacement
       6B 04   FD AD   0C  0D58  2494          PROBER  MODE(FP),#4,(R11)               ; can we read the displacement ?
                06   12  0D5D  2495          BNEQ    1$                              ; yes - skip
             5A 04   D0  0D5F  2496          MOVL    #4,R10                          ; R10 = size of probe
                0186   30  0D62  2497          BSBW    READ_FAULT                      ; process the access violation
          50 AD   04   C0  0D65  2498 1$:     ADDL2   #4,REG_PC(FP)                   ; increment PC
       0E 54 AD   1B   E0  0D69  2499          BBS     #PSL$V_FPD,PSL(FP),2$           ; skip if FPD set
             5B   6B   D0  0D6E  2500          MOVL    (R11),R11                       ; R11 = displacement value
             5B   57   C0  0D71  2501          ADDL2   R7,R11                          ; add the index to the displacement
          5B   14 AD42   C0  0D74  2502          ADDL2   REG_R0(FP)[R2],R11              ; add the register to the address
                0038   31  0D79  2503          BRW     ACCESS_VALUE                    ; finish establishing the access
                       05  0D7C  2504 2$:     RSB                                     ; Return if FPD set
                           0D7D  2505          ;
                           0D7D  2506          ;       Process a Long Displacement Deferred Mode Operand Specifier
                           0D7D  2507          ;
                           0D7D  2508 LONG_DEF_MODE:                                   ; entrance
          5B   50 AD   D0  0D7D  2509          MOVL    REG_PC(FP),R11                  ; R11 = location of the displacement
       6B 04   FD AD   0C  0D81  2510          PROBER  MODE(FP),#4,(R11)               ; can we read the displacement ?
                06   12  0D86  2511          BNEQ    1$                              ; yes - skip
             5A 04   D0  0D88  2512          MOVL    #4,R10                          ; R10 = size of probe
                015D   30  0D8B  2513          BSBW    READ_FAULT                      ; process the access violation
          50 AD   04   C0  0D8E  2514 1$:     ADDL2   #4,REG_PC(FP)                   ; increment PC
       1C 54 AD   1B   E0  0D92  2515          BBS     #PSL$V_FPD,PSL(FP),3$           ; skip if FPD set
             5B   6B   D0  0D97  2516          MOVL    (R11),R11                       ; R11 = displacement value
          5B   14 AD42   C0  0D9A  2517          ADDL2   REG_R0(FP)[R2],R11              ; add the register to the displacement
```

```
                6B   04    FD AD   0C  0D9F  2518          PROBER  MODE(FP),#4,(R11)       ; can we read longword it addresses ?
                           06  12   0C  0DA4  2519          BNEQ    2$                      ; yes - skip
                      5A   04  D0   0C  0DA6  2520          MOVL    #4,R10                  ; R10 = size of probe
                    013F   30  0C  0DA9  2521          BSBW    READ_FAULT              ; process the access violation
                5B  6B     57  C1   0C  0DAC  2522 2$:      ADDL3   R7,(R11),R11            ; form the operand address
                    0001   31  0C  0DB0  2523          BRW     ACCESS_VALUE            ; finish establishing the access
                           05  0C  0DB3  2524 3$:      RSB                             ; Return if FPD set
                               0C  0DB4  2525          ;
                               0C  0DB4  2526          ;       Set Up the Type of Access Requested
                               0C  0DB4  2527          ;
                               0C  0DB4  2528 ACCESS_VALUE:                           ; entrance
                04  01  58  CF   0C  0DB4  2529          CASEL   R8,#1,#4                ; branch on the access type
                         0016' 0C  0DB8  2530 1$:      .WORD   READ_CHECK-1$           ; 1 - read only access
                         000C' 0C  0DBA  2531          .WORD   MODIFY_CHECK-1$         ; 2 - modify access
                         007B' 0C  0DBC  2532          .WORD   WRITE_CHECK-1$          ; 3 - write only access
                         000B' 0C  0DBE  2533          .WORD   2$-1$                   ; 4 - address access
                         000B' 0C  0DC0  2534          .WORD   2$-1$                   ; 5 - field access
                           00  0C  0DC2  2535          HALT                            ; 6 - branch access (shouldn't occur)
                           05  0C  0DC3  2536 2$:      RSB                             ; return with the operand address
                               0C  0DC4  2537          ;
                               0C  0DC4  2538          ;       Perform Error Checking for Modify Access Operands
                               0C  0DC4  2539          ;
                               0C  0DC4  2540 MODIFY_CHECK:                           ; entrance
                           6D  10  0C  0DC4  2541          BSBB    WRITE_CHECK             ; check write (and read) access
                FE78 CD45  5B  D0   0C  0DC6  2542          MOVL    R11,READ_ADDRS(FP)[R5]  ; Set read operand address
                           1F  11  0C  0DCC  2543          BRB     ROPRAND_CHECK           ; Check for reserved operand
                               0C  0DCE  2544          ;
                               0C  0DCE  2545          ;       Perform Error Checking for Read Only Access Operands
                               0C  0DCE  2546          ;
                               0C  0DCE  2547 READ_CHECK:                             ; entrance
                FE78 CD45  5B  D0   0C  0DCE  2548          MOVL    R11,READ_ADDRS(FP)[R5]  ; Set read operand address
                     FE38 CD45 D4   0C  0DD4  2549          CLRL    WRITE_ADDRS(FP)[R5]     ; Indicate no write operand
                6B   5A    FD AD   0C  0DD9  2550          PROBER  MODE(FP),R10,(R11)      ; can we read the operand ?
                           0D  12   0C  0DDE  2551          BNEQ    ROPRAND_CHECK           ; yes - test for reserved operand
                               0C  0DE0  2552          CMPCOND SS$_ACCVIO,COND_NAME(FP); Is this an SS$_ACCVIO fault?
                           01  12   0C  0DE7  2553          BNEQ    1$                      ; Skip if not
                           05  0C  0DE9  2554          RSB                             ; Return
                      00FE 30  0C  0DEA  2555 1$:      BSBW    READ_FAULT              ; process the access violation
                               0C  0DED  2556 ROPRAND_CHECK:
                               0C  0DED  2557          CMPCOND SS$_ROPRAND,COND_NAME(FP); Is exception SS$_ROPRAND?
                           35  13   0C  0DF8  2558          BEQL    10$                     ; If so, skip operand check
                08  01  59  CF   0C  0DFA  2559          CASEL   R9,#1,#8                ; case on data type
                         0031' 0C  0DFE  2560 1$:      .WORD   10$-1$                  ; 1 - byte
                         0031' 0C  0E00  2561          .WORD   10$-1$                  ; 2 - word
                         0031' 0C  0E02  2562          .WORD   10$-1$                  ; 3 - longword
                         0031' 0C  0E04  2563          .WORD   10$-1$                  ; 4 - quadword
                         0031' 0C  0E06  2564          .WORD   10$-1$                  ; 5 - octaword
                         0012' 0C  0E08  2565          .WORD   2$-1$                   ; 6 - F_floating
                         0012' 0C  0E0A  2566          .WORD   2$-1$                   ; 7 - D_floating
                         001E' 0C  0E0C  2567          .WORD   3$-1$                   ; 8 - G_floating
                         002A' 0C  0E0E  2568          .WORD   4$-1$                   ; 9 - H_floating
00000100 8F     6B  09  07   ED   0C  0E10  2569 2$:      CMPZV   #7,#9,(R11),#^X100      ; F or D reserved operand?
                           15  13   0C  0E19  2570          BEQL    11$                     ; If so, SS$_ROPRAND
                               05   0C  0E1B  2571          RSB                             ; Else return
00000800 8F     6B  0C  04   ED   0C  0E1C  2572 3$:      CMPZV   #4,#12,(R11),#^X800     ; G reserved operand?
                           09  13   0C  0E25  2573          BEQL    11$                     ; If so, SS$_ROPRAND
                               05   0C  0E27  2574          RSB                             ; Else return
```

```
                                                                ;
                                                                ; Perform Error Checking for Write Only Access Operands
                                                                ;
                                                WRITE_CHECK:                            ; entrance
                6B   5A   FD AD   0D  0E33  2584        PROBEW   MODE(FP),R10,(R11)       ; can we write the operand ?
                               14   12  0E38  2585        BNEQ    1$                      ; yes - bypass
                                                0E3A  2586        CMPCOND SS$_ACCVIO,COND_NAME(FP); Is exception SS$_ACCVIO?
                               0B   13  0E41  2587        BEQL    1$                      ; If so, skip access test
                             0116   30  0E43  2588        BSBW    WRITE_FAULT             ; process the access violation
                  03 FC AD     00   E0  0E46  2589        BBS     #V_REGISTER,FLAGS(FP),1$ ; no local store checking - skip
                             0086   30  0E4B  2590        BSBW    LOCAL_TEST              ; test for a write into local storage
                       FE78 CD45   D4  0E4E  2591 1$:     CLRL    READ_ADDRS(FP)[R5]      ; Indicate no reading
                  FE38 CD45   5B   D0  0E53  2592        MOVL    R11,WRITE_ADDRS(FP)[R5] ; Indicate write operand address
                               05   RSB  0E59  2593
                                                0E5A  2594
                                                0E5A  2595 BRANCH_ACCESS:
                       FE38 CD45   D4  0E5A  2596        CLRL    WRITE_ADDRS(FP)[R5]     ; Indicate no write operand
                    02   01   59   CF  0E5F  2597        CASEL   R9,#1,#2                ; Case on data type
                                0009' 0E63  2598 1$:     .WORD   BRANCH_BYTE-1$          ; 1 - byte
                                002B' 0E65  2599         .WORD   BRANCH_WORD-1$          ; 2 - word
                                004E' 0E67  2600         .WORD   BRANCH_LONG-1$          ; 3 - longword
                             0149   31  0E69  2601        BRW     ADDRESS_FAULT           ; All other types get SS$_RADRMOD
                                                0E6C  2602
                                                0E6C  2603
                                                0E6C  2604      ;
                                                0E6C  2605      ; Process a Byte Branch Displacement Operand
                                                0E6C  2606      ;
                                                0E6C  2606 BRANCH_BYTE:                 ; entrance
                     5B   50 AD   D0  0E6C  2607        MOVL    REG_PC(FP),R11          ; R11 = location of the displacement
                6B   01   FD AD   0C  0E70  2608        PROBER  MODE(FP),#1,(R11)       ; can we read the displacement ?
                               06   12  0E75  2609        BNEQ    1$                      ; yes - skip
                          5A   01   D0  0E77  2610        MOVL    #1,R10                  ; R10 = size of probe
                             006E   30  0E7A  2611        BSBW    READ_FAULT              ; process the access violation
                          5B   6B   98  0E7D  2612 1$:     CVTBL   (R11),R11               ; R11 = branch displacement
                            50 AD   D6  0E80  2613        INCL    REG_PC(FP)              ; increment PC
                     5B     50 AD   C0  0E83  2614        ADDL2   REG_PC(FP),R11          ; compute the branch destination
                   FE78 CD45   5B   D0  0E87  2615        MOVL    R11,READ_ADDRS(FP)[R5]  ; Store the branch address
                               05   RSB  0E8D  2616                                        ; return
                                                0E8E  2617
                                                0E8E  2618      ;
                                                0E8E  2619      ; Process a Word Branch Displacement Operand
                                                0E8E  2620      ;
                                                0E8E  2621 BRANCH_WORD:                 ; entrance
                     5B   50 AD   D0  0E8E  2622        MOVL    REG_PC(FP),R11          ; R11 = location of the displacement
                6B   02   FD AD   0C  0E92  2623        PROBER  MODE(FP),#2,(R11)       ; can we read the displacement ?
                               06   12  0E97  2624        BNEQ    1$                      ; yes - skip
                          5A   02   D0  0E99  2625        MOVL    #2,R10                  ; R10 = size of probe
                             004C   30  0E9C  2626        BSBW    READ_FAULT              ; process the access violation
                          5B   6B   32  0E9F  2627 1$:     CVTWL   (R11),R11               ; R11 = branch displacement
                     50 AD   02     C0  0EA2  2628        ADDL2   #2,REG_PC(FP)           ; increment PC
                     5B     50 AD   C0  0EA6  2629        ADDL2   REG_PC(FP),R11          ; compute the branch destination
                   FE78 CD45   5B   D0  0EAA  2630        MOVL    R11,READ_ADDRS(FP)[R5]  ; Store the branch address
                               05   RSB  0EB0  2631                                        ; return
```

```
                        0EB1  2632
                        0EB1  2633          ;
                        0EB1  2634          ;       Process a Longword Branch Displacement Operand
                        0EB1  2635          ;
                        0EB1  2636 BRANCH_LONG:                                 ; entrance
      5B    50 AD   D0  0EB1  2637          MOVL    REG_PC(FP),R11              ; R11 = location of the displacement
   6B 04    FD AD   0C  0EB5  2638          PROBER  MODE(FP),#4,(R11)           ; can we read the displacement ?
               06   12  0EBA  2639          BNEQ    1$                          ; yes - skip
            5A  04   D0  0EBC  2640          MOVL    #4,R10                      ; R10 = size of probe
              0029   30  0EBF  2641          BSBW    READ_FAULT                  ; process the access violation
            5B  6B   D0  0EC2  2642 1$:     MOVL    (R11),R11                   ; R11 = branch displacement
         50 AD  04   C0  0EC5  2643          ADDL2   #4,REG_PC(FP)               ; increment PC
      5B    50 AD   C0  0EC9  2644          ADDL2   REG_PC(FP),R11              ; compute the branch destination
 FE78 CD45    5B   D0  0ECD  2645          MOVL    R11,READ_ADDRS(FP)[R5]      ; Store branch address
               05   05  0ED3  2646          RSB                                 ; return
                        0ED4  2647
                        0ED4  2648          ;
                        0ED4  2649          ;       Test for a Write into Local Storage
                        0ED4  2650          ;
                        0ED4  2651          ;               entered by subroutine branching
                        0ED4  2652          ;
                        0ED4  2653          ;               parameters:     R10 = Number of Bytes to be Written
                        0ED4  2654          ;                               R11 = Destination Address
                        0ED4  2655          ;
                        0ED4  2656          ;               returns with    R11 = Corrected Destination Address
                        0ED4  2657          ;
                        0ED4  2658          ;       Discussion
                        0ED4  2659          ;
                        0ED4  2660          ;           This routine checks the write operation described by
                        0ED4  2661          ;       the parameters in R10 and R11 for a write into the Emulator's
                        0ED4  2662          ;       working storage. If such a write is about to take place, R11
                        0ED4  2663          ;       is changed to an address where the write will not do any harm.
                        0ED4  2664          ;
                        0ED4  2665 LOCAL_TEST:                                  ; entrance
      53    5B AD   9E  0ED4  2666          MOVAB   LOCAL_END(FP),R3            ; R3 = byte following local storage
         53    5B   D1  0ED8  2667          CMPL    R11,R3                      ; is the write above the frame ?
               0D   1E  0EDB  2668          BGEQU   1$                          ; yes - bypass
   53    5B    5A   C1  0EDD  2669          ADDL3   R10,R11,R3                  ; R3 = byte following operand
         5E    53   D1  0EE1  2670          CMPL    R3,SP                       ; is it above the stack pointer ?
               04   1B  0EE4  2671          BLEQU   1$                          ; no - operand is not in local storage
      5B    5B AD   9E  0EE6  2672          MOVAB   TEMP(FP),R11                ; redirect the write to TEMP
               05   05  0EEA  2673 1$:     RSB                                 ; return with the operand address
                        0EEB  2674          ;
```

```
                        ;*****************************************************************
                        ;*                                                               *
                        ;*                                                               *
                        ;*                Exception Processing Routines                  *
                        ;*                                                               *
                        ;*                                                               *
                        ;*****************************************************************
                        ;
                        ;
                        ; Introduction
                        ; ------------
                        ;
                        ;    For each of the exceptions recognized, there is a routine
                        ; which is branched to (except for access violations in which
                        ; a subroutine branch is used instead) as soon as the condition
                        ; is detected. This routine pushes a shortened version of the
                        ; signal array onto the stack and branches to SIGNAL_START which
                        ; builds the signal and mechanism arrays in the proper place in
                        ; memory and enters the signal dispatcher to search for handlers
                        ; to process the condition. If the exception was a fault, the
                        ; routine FAULT_RESET is called to restore the registers to
                        ; their values when the instruction was started.
                        ;
                        ;
                        ; Access Violations
                        ; ----------------
                        ;
                        ;    The routines READ_FAULT and WRITE_FAULT are called by
                        ; subroutine branching when memory probes of read and write
                        ; access fail during instruction emulation. The register R11 is
                        ; assumed to contain the location of the area being probed
                        ; and the register R10 is assumed to contain its length. The
                        ; routine tries to produce the fault under controlled conditions
                        ; and returns if it can not produce the fault. If it can produce
                        ; the fault the the fault is signaled with the reason mask being
                        ; the reason mask from the attempt to produce the fault and with
                        ; the violation address as the address of the first byte of the
                        ; area for which the access violation occurs.
                        ;
                        ;
```

```
                     0EEB  2718          ;
                     0EEB  2719          ;       READ_FAULT - Process a Read Access Violation Fault
                     0EEB  2720          ;
                     0EEB  2721          ;               entered by subroutine branching
                     0EEB  2722          ;
                     0EEB  2723          ;               parameters:     R10 = Size of Area being Read
                     0EEB  2724          ;                               R11 = Location of Area being Read
                     0EEB  2725          ;
                     0EEB  2726 READ_FAULT:                              ; entrance
                  07 BB  0EFB  2727          PUSHR   #^M<R0,R1,R2>       ; save R0,R1,R2
                52 5B  D0  0EFD  2728          MOVL    R11,R2              ; R2 = probed address
          62    01 FD AD  0C  0F00  2729          PROBER  MODE(FP),#1,(R2)    ; is the first byte readable ?
                   13  13  0F05  2730          BEQL    1$                  ; no - bypass
      FF A24A   01 FD AD  0C  0F07  2731          PROBER  MODE(FP),#1,-1(R2)[R10] ; is the last byte readable ?
                   0A  12  0F0E  2732          BNEQ    1$                  ; yes - bypass
             50 FF A24A 9E  0F10  2733          MOVAB   -1(R2)[R10],R0      ; R2 = address of last byte
             52 01FF 8F AA  0F15  2734          BICW2   #511,R2             ; compute address of first bad byte
               26'AF 00 FB  0F1A  2735 1$:       CALLS   #0,B^READ_REASON    ; get the reason mask
                  12 50 E8  0F1E  2736          BLBS    R0,2$               ; the read went all right - bypass
                   012B 30  0F21  2737          BSBW    FAULT_RESET         ; reinitialize registers and clear TP
                5E F8 AD 9E  0F24  2738          MOVAB   SHORT_LOCAL(FP),SP  ; shorten the frame
                   52 DD  0F28  2739          PUSHL   R2                  ; push the bad address
                   51 DD  0F2A  2740          PUSHL   R1                  ; push the reason mask
                   0C DD  0F2C  2741          PUSHL   #SS$_ACCVIO         ; push the condition code
                   03 DD  0F2E  2742          PUSHL   #3                  ; push the number of arguments
                  014C 31  0F30  2743          BRW     SIGNAL_START        ; signal the condition
                   07 BA  0F33  2744 2$:       POPR    #^M<R0,R1,R2>       ; restore R0,R1,R2
                   05 05  0F35  2745          RSB                         ; get back
                      0F36  2746          ;
                      0F36  2747          ;       READ_REASON - Get the Reason Mask for a Read Access Violation
                      0F36  2748          ;
                      0F36  2749          ;               parameter:      R2 = Address for which Probe Failed
                      0F36  2750          ;
                      0F36  2751          ;               returns with    R0 = Status of Access Attempt
                      0F36  2752          ;                               R1 = Reason Mask if Unsuccessful
                      0F36  2753          ;
                      0F36  2754 READ_REASON:                             ; entrance
                0000  0F36  2755          .WORD   0                   ; entry mask
      6D  32'AF 9E  0F38  2756          MOVAB   B^REASON_HANDLER,(FP) ; set up the condition handler
             62 95  0F3C  2757          TSTB    (R2)                ; touch the location
          50 01 D0  0F3E  2758          MOVL    #1,R0               ; indicate a successful read
                04  0F41  2759          RET                         ; return
                      0F42  2760          ;
                      0F42  2761          ;       REASON_HANDLER - Condition Handler for Reason Routines
                      0F42  2762          ;
                      0F42  2763          ;               parameters:     P1 = Signal Array Location
                      0F42  2764          ;                               P2 = Mechanism Array Location
                      0F42  2765          ;
                      0F42  2766          ;               returns with    R0 = Condition Response
                      0F42  2767          ;
                      0F42  2768 REASON_HANDLER:                          ; entrance
                0000  0F42  2769          .WORD   0                   ; entry mask
      50  04 AC 7D  0F44  2770          MOVQ    4(AP),R0            ; R0 and R1 = location of arrays
          08 A1 D5  0F48  2771          TSTL    8(R1)               ; condition from establisher frame ?
             19 12  0F4B  2772          BNEQ    1$                  ; no - bypass
                      0F4D  2773          CMPCOND SS$_ACCVIO,4(R0)    ; access violation condition ?
             11 12  0F53  2774          BNEQ    1$                  ; no - bypass
```

```
                     0C A1   D4   0F45  2775          CLRL    12(R1)                  ; return zero status in R0
                10 A1   08 A0   D0   0F48  2776          MOVL    8(R0),16(R1)            ; return the reason mask in R1
                        7E   7C   0F4D  2777          CLRQ    -(SP)                   ; default PC and level for unwind
          00000000'GF    02   FB   0F4F  2778          CALLS   #2,G^SYS$UNWIND         ; unwind the reason routine frame
               50   0918 8F   32   0F56  2779 1$:       CVTWL   #SS$_RESIGNAL,R0        ; specify condition not handled
                          04   0F5B  2780               RET                             ; return
                               0F5C  2781               ;
```

```
                                0F5C  2783          ;
                                0F5C  2784          ;       WRITE_FAULT - Process a Write Access Violation Fault
                                0F5C  2785          ;
                                0F5C  2786          ;               entered by subroutine branching
                                0F5C  2787          ;
                                0F5C  2788          ;               parameters:     R10 = Size of Area being Written
                                0F5C  2789          ;                               R11 = Location of Area being Written
                                0F5C  2790          ;
                                0F5C  2791 WRITE_FAULT:                             ; entrance
                          07    BB    0F5C  2792          PUSHR   #^M<R0,R1,R2>     ; save R0,R1,R2
                       52 5B    D0    0F5E  2793          MOVL    R11,R2            ; R2 = probed address
                    62 01 FD AD 0D    0F61  2794          PROBEW  MODE(FP),#1,(R2)  ; is the first byte writeable ?
                          13    13    0F66  2795          BEQL    1$                ; no - bypass
                FF A24A 01 FD AD 0D    0F68  2796          PROBEW  MODE(FP),#1,-1(R2)[R10] ; is the last byte writeable ?
                          0A    12    0F6F  2797          BNEQ    1$                ; yes - bypass
                    52 FF A24A 9E    0F71  2798          MOVAB   -1(R2)[R10],R2    ; R2 = address of last byte
                    52 01FF 8F AA    0F76  2799          BICW2   #511,R2           ; compute address of first bad byte
                  97'AF 00    FB    0F7B  2800 1$:      CALLS   #0,B^WRITE_REASON ; get the reason mask
                       12 50 E8    0F7F  2801          BLBS    R0,2$             ; the write went all right - bypass
                          00BA 30    0F82  2802          BSBW    FAULT_RESET       ; reinitialize registers and clear TP
                    5E F8 AD 9E    0F85  2803          MOVAB   SHORT_LOCAL(FP),SP ; shorten the frame
                          52    DD    0F89  2804          PUSHL   R2                ; push the bad address
                          51    DD    0F8B  2805          PUSHL   R1                ; push the reason mask
                          0C    DD    0F8D  2806          PUSHL   #SS$_ACCVIO       ; push the condition code
                          03    DD    0F8F  2807          PUSHL   #3                ; push the number of arguments
                          00DB 31    0F91  2808          BRW     SIGNAL_START      ; signal the condition
                          07    BA    0F94  2809 2$:      POPR    #^M<R0,R1,R2>     ; restore R0,R1,R2
                          05    0F96  2810          RSB                       ; get back
                                0F97  2811          ;
                                0F97  2812          ;       WRITE_REASON - Get the Reason Mask for Write Access Violation
                                0F97  2813          ;
                                0F97  2814          ;               parameter:      R2 = Address for which Probe Failed
                                0F97  2815          ;
                                0F97  2816          ;               returns with    R0 = Status of Access Attempt
                                0F97  2817          ;                               R1 = Reason Mask if Unsuccessful
                                0F97  2818          ;
                                0F97  2819 WRITE_REASON:                            ; entrance
                          0000  0F97  2820          .WORD   0                 ; entry mask
                6D 96 AF    9E    0F99  2821          MOVAB   B^REASON_HANDLER,(FP) ; set up the condition handler
                   62 00    80    0F9D  2822          ADDB2   #0,(R2)           ; try to change the location
                   50 01    D0    0FA0  2823          MOVL    #1,R0             ; indicate a successful write
                          04    0FA3  2824          RET                       ; return
                                0FA4  2825          ;
```

```
                        0FA4  2827          ;
                        0FA4  2828          ;       OPCODE_FAULT - Process an Opcode Reserved to Digital Fault
                        0FA4  2829          ;
                        0FA4  2830          ;               entered by branching
                        0FA4  2831          ;
                        0FA4  2832          ;               no parameters
                        0FA4  2833          ;
                        0FA4  2834 OPCODE_FAULT:                                ; entrance
          0098    30    0FA4  2835          BSBW    FAULT_RESET                 ; reinitialize registers and clear TP
     5E    F8 AD    9E  0FA7  2836          MOVAB   SHORT_LOCAL(FP),SP          ; shorten the frame
  7E    043C 8F    3C  0FAB  2837          MOVZWL  #SS$_OPCDEC,-(SP)           ; push the condition code
          01    DD  0FB0  2838          PUSHL   #1                              ; push the number of arguments
          00BA  31  0FB2  2839          BRW     SIGNAL_START                    ; signal the condition
                        0FB5  2840          ;
                        0FB5  2841          ;       ADDRESS_FAULT - Process an Invalid Addressing Mode Fault
                        0FB5  2842          ;
                        0FB5  2843          ;               entered by branching
                        0FB5  2844          ;
                        0FB5  2845          ;               no parameters
                        0FB5  2846          ;
                        0FB5  2847 ADDRESS_FAULT:                               ; entrance
          0087    30    0FB5  2848          BSBW    FAULT_RESET                 ; reinitialize registers and clear TP
     5E    F8 AD    9E  0FB8  2849          MOVAB   SHORT_LOCAL(FP),SP          ; shorten the frame
  7E    044C 8F    3C  0FBC  2850          MOVZWL  #SS$_RADRMOD,-(SP)          ; push the condition code
          01    DD  0FC1  2851          PUSHL   #1                              ; push the number of arguments
          00A9  31  0FC3  2852          BRW     SIGNAL_START                    ; signal the condition
                        0FC6  2853          ;
                        0FC6  2854          ;       OPERAND_FAULT - Processed a Reserved Operand Fault
                        0FC6  2855          ;
                        0FC6  2856          ;               entered by branching
                        0FC6  2857          ;
                        0FC6  2858          ;               no parameters
                        0FC6  2859          ;
                        0FC6  2860 OPERAND_FAULT:                               ; entrance
          0076    30    0FC6  2861          BSBW    FAULT_RESET                 ; reinitialize registers and clear TP
     5E    F8 AD    9E  0FC9  2862          MOVAB   SHORT_LOCAL(FP),SP          ; shorten the frame
  7E    0454 8F    3C  0FCD  2863          MOVZWL  #SS$_ROPRAND,-(SP)          ; push the condition code
          01    DD  0FD2  2864          PUSHL   #1                              ; push the number of arguments
          0098  31  0FD4  2865          BRW     SIGNAL_START                    ; signal the condition
```

```
                     0FD7  2867          ;       RESIGNAL - Resignal original exception
                     0FD7  2868          ;
                     0FD7  2869          ;               entered by branching
                     0FD7  2870          ;
                     0FD7  2871          ;               no parameters
                     0FD7  2872          ;
                     0FD7  2873          ;       This routine is branched to when the user's action routine
                     0FD7  2874          ;       returns to us with a failure status, indicating the desire
                     0FD7  2875          ;       to resignal the original exception.  First we call
                     0FD7  2876          ;       FAULT_RESET to undo any register modifications caused by
                     0FD7  2877          ;       operand processing.  Note that we assume that the user has
                     0FD7  2878          ;       NOT modified any register or the PSL before requesting
                     0FD7  2879          ;       a resignal.  We then reset our SP to point to the original
                     0FD7  2880          ;       signal array (it's probably already there anyway) and
                     0FD7  2881          ;       decrement the parameter count by 2, as needed by
                     0FD7  2882          ;       SIGNAL_START.  We then branch to SIGNAL_START.
                     0FD7  2883          ;
                     0FD7  2884
                     0FD7  2885 RESIGNAL:
            66   10   0FD7  2886          BSBB    FAULT_RESET                     ; Undo register modifications
     5E FDDC CD   D0   0FD9  2887          MOVL    SAVE_SIGARGS(FP),SP             ; Point SP to signal args
        6E    02   C2   0FDE  2888          SUBL2   #2,(SP)                         ; Adjust argument count.
      FC AD    02   88   0FE1  2889          BISB2   #M_RESIGNAL,FLAGS(FP)           ; Indicate a resignal
          0087   31   0FE5  2890          BRW     SIGNAL_START                    ; Do the signal
```

```
                     ;
                     ; USER_SIGNAL - Signal user-supplied exception
                     ;
                     ; Calling sequence:
                     ;
                     ;   CALL USER_SIGNAL (fault_flag.rl.r, context.rl.r,
                     ;                     signal_args.rl.ra)
                     ;
                     ; Parameters:
                     ;
00000004             ;    fault_flag = 4       ; The address of a longword whose
                     ;                         ; low bit, if set, indicates that
                     ;                         ; this exception is to be signalled
                     ;                         ; as a fault.  If the low bit is
                     ;                         ; clear, the exception is to be
                     ;                         ; signalled as a trap.
                     ;                         ;
                     ;                         ; If a fault, all register
                     ;                         ; modifications which resulted from
                     ;                         ; operand processing are rolled back.
                     ;                         ; If a trap, the current contents
                     ;                         ; of the registers are used.  The
                     ;                         ; current state of the PSL is used in
                     ;                         ; either case.
                     ;
00000008             ;   context = 8           ; The longword passed
                     ;                         ; to the user action routine as
                     ;                         ; the signal routine's context.
                     ;                         ; This longword contains the FP of
                     ;                         ; the appropriate invocation of
                     ;                         ; LIB$DECODE_FAULT.
                     ;
0000000C             ;           signal_args = 12  ; The address of an array of longwords
                     ;                         ; specifying the signal arguments of
                     ;                         ; the exception to be signalled.
                     ;                         ; The first longword contains the
                     ;                         ; count of following longwords.
                     ;                         ; Unlike the signal arguments list
                     ;                         ; passed to a condition handler, this
                     ;                         ; array does not contain the PC and
                     ;                         ; PSL, and the count reflects their
                     ;                         ; absence.
                     ;
                     ;       This routine is called from the user action routine when it wishes
                     ;       to signal an exception.  The address of this routine's entry mask
                     ;       was passed to the user action routine as an argument.
                     ;
                     ;       USER_SIGNAL unwinds the stack frames back to the frame of the
                     ;       associated invocation of LIB$DECODE_FAULT, whose FP was specified
                     ;       as "context".  This unwinding is not a full unwind, but only
                     ;       calls associated handlers with the SS$_UNWIND condition.
                     ;
                     ;       The user-specified signal args are then pushed on the stack and
                     ;       control branches to SIGNAL_START.  Depending on whether the
                     ;       "fault_flag" parameter is set, FAULT_RESET may or may not be called.
                     ;
                     ;
                     USER_SIGNAL:
```

```
                 001C  0FE8  2949          .WORD   ^M<R2,R3,R4>
   53    08 AC   D0  0FEA  2950          MOVL    context(AP),R3          ; Get context FP value
   54    04 BC   D0  0FEE  2951          MOVL    @fault_flag(AP),R4      ; Get fault indicator
   50    0C AC   D0  0FF2  2952          MOVL    signal_args(AP),R0      ; Get address of signal args
      51    60   D0  0FF6  2953          MOVL    (R0),R1                 ; Get argument count
50    04 A041   DE  0FF9  2954          MOVAL   4(R0)[R1],R0            ; Position R0 past signal args
         70     DD  0FFE  2955 1$:       PUSHL   -(R0)                   ; Push a signal argument
      FB 51     F4  1000  2956          SOBGEQ  R1,1$                   ; Loop till all pushed
                    1003  2957
                    1003  2958 ;+
                    1003  2959 ; Call handlers between our caller's frame and "context" frame with SS$_UNWIND
                    1003  2960 ;-
   52    0C AD   D0  1003  2961          MOVL    SF$L_SAVE_FP(FP),R2     ; Get caller's frame
         62     D5  1007  2962 2$:       TSTL    (R2)                    ; Does this frame have a handler?
         21     13  1009  2963          BEQL    3$                      ; Skip if not
         7E     7C  100B  2964          CLRQ    -(SP)                   ; Construct mechanism list
         7E     D4  100D  2965          CLRL    -(SP)                   ; Depth=0
         52     DD  100F  2966          PUSHL   R2                      ; Establisher's FP
         04     DD  1011  2967          PUSHL   #4                      ; n of mechanism args
7E    0920    8F  3C  1013  2968          MOVZWL  #SS$_UNWIND,-(SP)       ; Push unwind condition code
         01     DD  1018  2969          PUSHL   #1                      ; n of signal args
      08 AE     9F  101A  2970          PUSHAB  8(SP)                   ; Address of mechanism args
      04 AE     9F  101D  2971          PUSHAB  4(SP)                   ; Address of signal args
      51   62   D0  1020  2972          MOVL    (R2),R1                 ; Handler address in R1
00000000'GF    16  1023  2973          JSB     G^SYS$CALL_HANDL        ; Call the handler
      5E     24  C0  1029  2974          ADDL2   #36,SP                  ; Reset SP
   52    0C A2   D0  102C  2975 3$:       MOVL    SF$L_SAVE_FP(R2),R2     ; Get next frame
      53   52   D1  1030  2976          CMPL    R2,R3                   ; Is this our frame?
         D2     12  1033  2977          BNEQ    2$                      ; Loop if not
                    1035  2978
                    1035  2979 ;+
                    1035  2980 ; R2 now has the desired FP.  Switch frames.  SP is pointing to the signal
                    1035  2981 ; args.
                    1035  2982 ;-
                    1035  2983
5D    52    D0  1035  2984          MOVL    R2,FP                   ; Switch frames
   02 54    E9  1038  2985          BLBC    R4,4$                   ; Skip if not fault
      02    10  103B  2986          BSBB    FAULT_RESET             ; Roll back register modifications
      30    11  103D  2987 4$:      BRB     SIGNAL_START            ; Go do the signal
                103F  2988
```

```
                        103F  2990          ;
                        103F  2991          ;       FAULT_RESET - Perform Reinitialization Operations for a Fault
                        103F  2992          ;
                        103F  2993          ;               entered by subroutine branching
                        103F  2994          ;
                        103F  2995          ;               no parameters
                        103F  2996          ;
                        103F  2997          ;       Discussion
                        103F  2998          ;
                        103F  2999          ;           This routine restores the original register contents
                        103F  3000          ;       for a fault.
                        103F  3001          ;
                        103F  3002          ;
                        103F  3003  FAULT_RESET:
20 54 AD    1B  E0      103F  3004          BBS     #PSL$V_FPD,PSL(FP),1$   ; skip if FPD set
   50   B8 AD  9E      1044  3005          MOVAB   ORIG_R0(FP),R0          ; Address of original R0
   51   14 AD  9E      1048  3006          MOVAB   REG_R0(FP),R1           ; Address of place to restore R0
      81   80  7D      104C  3007          MOVQ    (R0)+,(R1)+             ; Restore R0-R1
      81   80  7D      104F  3008          MOVQ    (R0)+,(R1)+             ; Restore R2-R3
      81   80  7D      1052  3009          MOVQ    (R0)+,(R1)+             ; Restore R4-R5
      81   80  7D      1055  3010          MOVQ    (R0)+,(R1)+             ; Restore R6-R7
      81   80  7D      1058  3011          MOVQ    (R0)+,(R1)+             ; Restore R8-R9
      81   80  7D      105B  3012          MOVQ    (R0)+,(R1)+             ; Restore R10-R11
      81   80  7D      105E  3013          MOVQ    (R0)+,(R1)+             ; Restore AP-FP
      81   80  D0      1061  3014          MOVL    (R0)+,(R1)+             ; Restore SP
50 AD    F4 AD  D0      1064  3015  1$:     MOVL    ORIG_PC(FP),REG_PC(FP)  ; Restore PC
00 54 AD    1E  E5      1069  3016          BBCC    #PSL$V_TP,PSL(FP),2$    ; clear the trace pending bit
                05      106E  3017  2$:     RSB                             ; return
                        106F  3018          ;
```

```
                                ;
                                ;       SIGNAL_START - Build the Parameter Blocks for SIGNAL
                                ;
                                ;               entered by branching
                                ;
                                ;               parameters:     (SP) = Truncated Signal Array Size (M)
                                ;                               4(SP) = Condition Code
                                ;                               8(SP) = First Signal Argument
                                ;                                 .
                                ;                                 .
                                ;                                 .
                                ;                               4*<M-1>(SP) = Last Signal Argument
                                ;
                                ;       Discussion
                                ;
                                ;           This routine builds the signal and mechanism arrays
                                ;       for a condition generated by LIB$DECODE_FAULT. It is entered
                                ;       with the signal array for the condition except for the
                                ;       PC and PSL pair pushed onto the stack (with the
                                ;       pushed array length correspondingly shortened). The signal
                                ;       array, mechanism array, and the handler parameter block
                                ;       are then constructed on the user's emulated stack. The routine
                                ;       then removes LIB$DECODE_FAULT's frame from the stack and enters
                                ;       the signal dispatching loop at SIGNAL.
                                ;
                                ;       Notes:  1. The precise format of the information pushed onto
                                ;                  the user's stack is given in the description of
                                ;                  SIGNAL below.
                                ;
                                ;               2. The method of getting out of LIB$DECODE_FAULT used in
                                ;                  this routine is essentially the same as that used
                                ;                  in NORMAL_EXIT.
                                ;
                                SIGNAL_START:                                   ; entrance
      57    8E    D0  106F  3054            MOVL    (SP)+,R7                  ; R7 = number of signal parameters
58    57    02    78  1072  3055            ASHL    #2,R7,R8                  ; R8 = size of the signal parameters
50    58    34    C1  1076  3056            ADDL3   #52,R8,R0                 ; R0 = size of signal information
         F31B    30  107A  3057            BSBW    TEST_FRAME                ; make sure we have room for it
   56    4C AD    D0  107D  3058            MOVL    REG_SP(FP),R6             ; R6 = user's stack pointer
   76    50 AD    7D  1081  3059            MOVQ    REG_PC(FP),-(R6)          ; push the PC, PSL pair
      56    58    C2  1085  3060            SUBL2   R8,R6                     ; make room for the signal parameters
66    6E    58    28  1088  3061            MOVC3   R8,(SP),(R6)              ; push the signal parameters
76    57    02    C1  108C  3062            ADDL3   #2,R7,-(R6)               ; push the signal array length
      76    01    D0  1090  3063            MOVL    #1,-(R6)                  ; push code for SIGNAL (vs. STOP)
   76    14 AD    7D  1093  3064            MOVQ    REG_R0(FP),-(R6)          ; push user's R0 and R1
05 FC AD    01    E0  1097  3065            BBS     #V_RESIGNAL,FLAGS(FP),1$  ; Is this a resignal?
      76    03    CE  109C  3066            MNEGL   #3,-(R6)                  ; No, push -3 (depth number)
            04    11  109F  3067            BRB     2$                        ; Skip
   76    F8 AD    D0  10A1  3068 1$:        MOVL    SAVE_DEPTH(FP),-(R6)      ; Resignal, use saved handler depth
   76    48 AD    D0  10A5  3069 2$:        MOVL    REG_FP(FP),-(R6)          ; push the user's FP
      76    04    D0  10A9  3070            MOVL    #4,-(R6)                  ; push the mechanism array length
      76    56    D0  10AC  3071            MOVL    R6,-(R6)                  ; push the mechanism array location
   76    1C A6    9E  10AF  3072            MOVAB   28(R6),-(R6)              ; push the signal array location
      76    02    D0  10B3  3073            MOVL    #2,-(R6)                  ; push the handler parameter count
08 AD    44 AD    7D  10B6  3074            MOVQ    REG_AP(FP),SAVE_AP(FP)    ; put the user's PC, PSL pair back
10 AD    DC'AF    9E  10BB  3075            MOVAB   B^SIGNAL,SAVE_PC(FP)      ; store the return point
   50    48 AD    9E  10C0  3076            MOVAB   FRAME_END+4(FP),R0        ; R0 = location of end of frame
```

```
                    56   50   C2  10C4  3077          SUBL2   R0,R6                          ; R6 = distance of user SP from it
              51    56   02   00  EF  10C7  3078       EXTZV   #0,#2,R6,R1                    ; R1 = stack alignment
           06 AD    02   0E   51  F0  10CC  3079       INSV    R1,#MASK_ALIGN,#2,SAVE_MASK(FP) ; store it into the frame
                    50   51   C0  10D2  3080          ADDL2   R1,R0                          ; compute the parameter area location
             FC A0   56   FE 8F  78  10D5  3081       ASHL    #-2,R6,-4(R0)                  ; store the parameter count
                                 04  10DB  3082       RET                                    ; return (to SIGNAL)
                                     10DC  3083       ;
```

```
                     10DC  3085          ;
                     10DC  3086          ;  SIGNAL - Signal the Condition
                     10DC  3087          ;
                     10DC  3088          ;          entered by branching
                     10DC  3089          ;
                     10DC  3090          ;          parameters:     ( Described in Note 3 )
                     10DC  3091          ;
                     10DC  3092          ;  Discussion
                     10DC  3093          ;
                     10DC  3094          ;             Following is a description of the information which
                     10DC  3095          ;             is assumed to be pushed onto the stack when the
                     10DC  3096          ;             routine SIGNAL is entered. The values are all
                     10DC  3097          ;             longwords.
                     10DC  3098          ;
                     10DC  3099          ;             Handler Parameter Block:
                     10DC  3100          ;
                     10DC  3101          ;                  (SP)    2 (handler parameter block length)
                     10DC  3102          ;                  4(SP)   signal array location
                     10DC  3103          ;                  8(SP)   mechanism array location
                     10DC  3104          ;
                     10DC  3105          ;             Mechanism Array:
                     10DC  3106          ;
                     10DC  3107          ;                  12(SP)  4 (mechanism array length)
                     10DC  3108          ;                  16(SP)  user's FP (establisher frame)
                     10DC  3109          ;                  20(SP)  -3 (establisher depth)
                     10DC  3110          ;                  24(SP)  user's R0
                     10DC  3111          ;                  28(SP)  user's R1
                     10DC  3112          ;
                     10DC  3113          ;             Information Not Part of any Array:
                     10DC  3114          ;
                     10DC  3115          ;                  32(SP)  1 (code for SIGNAL)
                     10DC  3116          ;
                     10DC  3117          ;             Signal Array:
                     10DC  3118          ;
                     10DC  3119          ;                  36(SP)  signal array length M
                     10DC  3120          ;                  40(SP)  condition code
                     10DC  3121          ;                  44(SP)  first signal argument
                     10DC  3122          ;                    .
                     10DC  3123          ;                    .
                     10DC  3124          ;                    .
                     10DC  3125          ;                  <4*M>+28(SP) last signal argument
                     10DC  3126          ;                  <4*M>+32(SP) user's PC
                     10DC  3127          ;                  <4*M>+36(SP) user's PSL
                     10DC  3128          ;
                     10DC  3129          ;             The user's stack pointer should coincide with the
                     10DC  3130          ;             address <4*M>+40(SP).
                     10DC  3131          ;
                     10DC  3132          ;          We now jump to the VMS entry point to look for a handler.
                     10DC  3133          ;-
                     10DC  3134
                     10DC  3135 SIGNAL:
00000000'GF    17    10DC  3136          JMP     G^SYS$SRCHANDLER
                     10E2  3137
                     10E2  3138          .END                            ; End of module LIB$DECODE_FAULT
```

```
ACCESS_VALUE                    00000DB4 R      02
ADDRESS_FAULT                   00000FB5 R      02
BRANCH_ACCESS                   00000E5A R      02
BRANCH_BYTE                     00000E6C R      02
BRANCH_LONG                     00000EB1 R      02
BRANCH_WORD                     00000E8E R      02
BYTE_DEF_MODE                   00000CBE R      02
BYTE_DISP_MODE                  00000C96 R      02
CALL_ARGS                     = 00000050
CHF$C_MCH_DEPTH               = 00000008
CHF$L_MCH_SAVR0               = 0000000C
CHF$L_SIG_NAME                = 00000004
COND_HANDLER                    00000415 R      02
COND_NAME                     = FFFFFDD8
CONTEXT                       = 00000008
DECODE_FAULT                    00000211 R      02
DECR_MODE                       00000C1C R      02
DSC$A_POINTER                 = 00000004
DSC$B_DTYPE                   = 00000002
DSC$K_DTYPE_BPV               = 00000020
FAULT_FLAG                    = 00000004
FAULT_RESET                     0000103F R      02
FIELD_REG                       00000BB4 R      02
FLAGS                         = FFFFFFFC
FRAME_END                     = 00000044
GET_SPECIFIER                   00000A59 R      02
HANDLER                       = 00000000
INCR_DEF_MODE                   00000C69 R      02
INCR_MODE                       00000C38 R      02
INDEX_MODE                      00000B12 R      02
INSTR_DEF                     = FFFFFDEC
INSTR_FOUND                     0000013A R      02
INSTR_OPCODE                  = FFFFFDF0
INVALID_TYPE                    000002FA R      02
LAST_OPERAND                    00000307 R      02
LENGTHS                         00000A50 R      02
LIB$DECODE_FAULT                00000000 RG     02
LIB$GET_OPCODE                  ********   X    00
LIB$K_DCFACC_A                = 00000004
LIB$K_DCFACC_B                = 00000006
LIB$K_DCFACC_M                = 00000002
LIB$K_DCFACC_R                = 00000001
LIB$K_DCFACC_V                = 00000005
LIB$K_DCFACC_W                = 00000003
LIB$K_DCFOPR_END              = 00000000
LIB$K_DCFTYP_B                = 00000001
LIB$K_DCFTYP_D                = 00000007
LIB$K_DCFTYP_F                = 00000006
LIB$K_DCFTYP_G                = 00000008
LIB$K_DCFTYP_H                = 00000009
LIB$K_DCFTYP_L                = 00000003
LIB$K_DCFTYP_O                = 00000005
LIB$K_DCFTYP_Q                = 00000004
LIB$K_DCFTYP_W                = 00000002
LIB$STOP                        ********   X    00
LIB$S_DCFACC                  = 00000003
LIB$S_DCFTYP                  = 00000005
LIB$V_DCFACC                  = 00000000
LIB$V_DCFTYP                  = 00000003
LIB$_INVARG                     ********   X    00
LIB$_RESTART                    ********   X    00
LITERAL_MODE                    00000AAE R      02
LOCAL_END                     = 00000058
LOCAL_START                   = FFFFFDD8
LOCAL_TEST                      00000ED4 R      02
LONG_DEF_MODE                   00000D7D R      02
LONG_DISP_MODE                  00000D54 R      02
MASK_ALIGN                    = 0000000E
MECHARGS                      = 00000008
MODE                          = FFFFFFFD
MODIFY_CHECK                    00000DC4 R      02
MODIFY_REG                      00000BC1 R      02
M_REGISTER                    = 00000001
M_RESIGNAL                    = 00000002
NORMAL_EXIT                     00000351 R      02
NOT_FOUND                       00000134 R      02
N_OF_OPERANDS                 = FFFFFDF4
OPCODE_FAULT                    00000FA4 R      02
OPCODE_TABLE                  = 00000014
OPERAND_FAULT                   00000FC6 R      02
OPERAND_LOOP                    000002D0 R      02
OPERAND_TYPES                 = FFFFFDF8
ORIG_AP                       = FFFFFFE8
ORIG_FP                       = FFFFFFEC
ORIG_PC                       = FFFFFFF4
ORIG_R0                       = FFFFFFB8
ORIG_R1                       = FFFFFFBC
ORIG_R10                      = FFFFFFE0
ORIG_R11                      = FFFFFFE4
ORIG_R2                       = FFFFFFC0
ORIG_R3                       = FFFFFFC4
ORIG_R4                       = FFFFFFC8
ORIG_R5                       = FFFFFFCC
ORIG_R6                       = FFFFFFD0
ORIG_R7                       = FFFFFFD4
ORIG_R8                       = FFFFFFD8
ORIG_R9                       = FFFFFFDC
ORIG_SP                       = FFFFFFF0
PATRN_ACBB                      0000095D R      02
PATRN_ACBD                      00000906 R      02
PATRN_ACBF                      000008CA R      02
PATRN_ACBG                      000009ED R      02
PATRN_ACBH                      00000A20 R      02
PATRN_ACBL                      000009AB R      02
PATRN_ACBW                      000008A7 R      02
PATRN_ADAWI                     000008E4 R      02
PATRN_ADDB2                     0000093C R      02
PATRN_ADDB3                     0000093F R      02
PATRN_ADDD2                     000008ED R      02
PATRN_ADDD3                     000008F0 R      02
PATRN_ADDF2                     000008B1 R      02
PATRN_ADDF3                     000008B4 R      02
PATRN_ADDG2                     000009D4 R      02
PATRN_ADDG3                     000009D7 R      02
```

PATRN_ADDH2	00000A07	R	02	PATRN_BVS	00000862	R	02				
PATRN_ADDH3	00000A0A	R	02	PATRN_CALLG	000009CB	R	02				
PATRN_ADDL2	00000979	R	02	PATRN_CALLS	000009C8	R	02				
PATRN_ADDL3	0000097C	R	02	PATRN_CASEB	00000946	R	02				
PATRN_ADDP4	00000866	R	02	PATRN_CASEL	00000983	R	02				
PATRN_ADDP6	0000086B	R	02	PATRN_CASEW	0000096C	R	02				
PATRN_ADDW2	000008E4	R	02	PATRN_CHME	00000975	R	02				
PATRN_ADDW3	00000965	R	02	PATRN_CHMK	00000975	R	02				
PATRN_ADWC	00000979	R	02	PATRN_CHMS	00000975	R	02				
PATRN_AOBLEQ	000009B0	R	02	PATRN_CHMU	00000975	R	02				
PATRN_AOBLSS	000009B0	R	02	PATRN_CLRB	0000094D	R	02				
PATRN_ASHL	00000920	R	02	PATRN_CLRL	0000098A	R	02				
PATRN_ASHP	000009BD	R	02	PATRN_CLRO	00000A3A	R	02				
PATRN_ASHQ	00000924	R	02	PATRN_CLRQ	00000932	R	02				
PATRN_BBC	00000995	R	02	PATRN_CLRW	00000973	R	02				
PATRN_BBCC	00000995	R	02	PATRN_CMPB	0000094A	R	02				
PATRN_BBCCI	00000995	R	02	PATRN_CMPC3	00000878	R	02				
PATRN_BBCS	00000995	R	02	PATRN_CMPC5	00000881	R	02				
PATRN_BBS	00000995	R	02	PATRN_CMPD	0000090E	R	02				
PATRN_BBSC	00000995	R	02	PATRN_CMPF	000008D2	R	02				
PATRN_BBSS	00000995	R	02	PATRN_CMPG	000009F5	R	02				
PATRN_BBSSI	00000995	R	02	PATRN_CMPH	00000A28	R	02				
PATRN_BEQL	00000862	R	02	PATRN_CMPL	00000987	R	02				
PATRN_BGEQ	00000862	R	02	PATRN_CMPP3	00000896	R	02				
PATRN_BGEQU	00000862	R	02	PATRN_CMPP4	00000866	R	02				
PATRN_BGTR	00000862	R	02	PATRN_CMPV	000009A1	R	02				
PATRN_BGTRU	00000862	R	02	PATRN_CMPW	00000970	R	02				
PATRN_BICB2	0000093C	R	02	PATRN_CMPZV	000009A1	R	02				
PATRN_BICB3	0000093F	R	02	PATRN_CRC	00000853	R	02				
PATRN_BICL2	00000979	R	02	PATRN_CVTBD	000008FD	R	02				
PATRN_BICL3	0000097C	R	02	PATRN_CVTBF	000008C1	R	02				
PATRN_BICPSW	00000975	R	02	PATRN_CVTBG	000009E4	R	02				
PATRN_BICW2	000008E4	R	02	PATRN_CVTBH	00000A17	R	02				
PATRN_BICW3	00000965	R	02	PATRN_CVTBL	00000953	R	02				
PATRN_BISB2	0000093C	R	02	PATRN_CVTBW	00000956	R	02				
PATRN_BISB3	0000093F	R	02	PATRN_CVTDB	000008F4	R	02				
PATRN_BISL2	00000979	R	02	PATRN_CVTDF	0000091D	R	02				
PATRN_BISL3	0000097C	R	02	PATRN_CVTDH	000009CE	R	02				
PATRN_BISPSW	00000975	R	02	PATRN_CVTDL	000008FA	R	02				
PATRN_BISW2	000008E4	R	02	PATRN_CVTDW	000008F7	R	02				
PATRN_BISW3	00000965	R	02	PATRN_CVTFB	000008B8	R	02				
PATRN_BITB	0000094A	R	02	PATRN_CVTFD	000008E1	R	02				
PATRN_BITL	00000987	R	02	PATRN_CVTFG	00000A47	R	02				
PATRN_BITW	00000970	R	02	PATRN_CVTFH	00000A44	R	02				
PATRN_BLBC	00000999	R	02	PATRN_CVTFL	000008BE	R	02				
PATRN_BLBS	00000999	R	02	PATRN_CVTFW	000008BB	R	02				
PATRN_BLEQ	00000862	R	02	PATRN_CVTGB	000009DB	R	02				
PATRN_BLEQU	00000862	R	02	PATRN_CVTGF	000009D1	R	02				
PATRN_BLSS	00000862	R	02	PATRN_CVTGH	00000A04	R	02				
PATRN_BLSSU	00000862	R	02	PATRN_CVTGL	000009E1	R	02				
PATRN_BNEQ	00000862	R	02	PATRN_CVTGW	000009DE	R	02				
PATRN_BPT	00000846	R	02	PATRN_CVTHB	00000A0E	R	02				
PATRN_BRB	00000862	R	02	PATRN_CVTHD	00000A4D	R	02				
PATRN_BRW	0000088E	R	02	PATRN_CVTHF	00000A4A	R	02				
PATRN_BSBB	00000862	R	02	PATRN_CVTHG	00000A37	R	02				
PATRN_BSBW	0000088E	R	02	PATRN_CVTHL	00000A14	R	02				
PATRN_BVC	00000862	R	02	PATRN_CVTHW	00000A11	R	02				

PATRN_CVTLB	000009B7	R	02	PATRN_INSQTI	000008E7	R	02
PATRN_CVTLD	00000903	R	02	PATRN_INSQUE	0000085C	R	02
PATRN_CVTLF	000008C7	R	02	PATRN_INSV	000009A6	R	02
PATRN_CVTLG	000009EA	R	02	PATRN_JMP	00000864	R	02
PATRN_CVTLH	00000A1D	R	02	PATRN_JSB	00000864	R	02
PATRN_CVTLP	000009C4	R	02	PATRN_LDPCTX	00000846	R	02
PATRN_CVTLW	000009BA	R	02	PATRN_LOCC	000008A3	R	02
PATRN_CVTPL	0000089A	R	02	PATRN_MATCHC	00000866	R	02
PATRN_CVTPS	00000847	R	02	PATRN_MCOMB	00000943	R	02
PATRN_CVTPT	00000872	R	02	PATRN_MCOML	00000980	R	02
PATRN_CVTRDL	000008FA	R	02	PATRN_MCOMW	00000969	R	02
PATRN_CVTRFL	000008BE	R	02	PATRN_MFPR	00000980	R	02
PATRN_CVTRGL	000009E1	R	02	PATRN_MNEGB	00000943	R	02
PATRN_CVTRHL	00000A14	R	02	PATRN_MNEGD	0000090B	R	02
PATRN_CVTSP	00000847	R	02	PATRN_MNEGF	000008CF	R	02
PATRN_CVTTP	00000872	R	02	PATRN_MNEGG	000009F2	R	02
PATRN_CVTWB	00000893	R	02	PATRN_MNEGH	00000A25	R	02
PATRN_CVTWD	00000900	R	02	PATRN_MNEGL	00000980	R	02
PATRN_CVTWF	000008C4	R	02	PATRN_MNEGW	00000969	R	02
PATRN_CVTWG	000009E7	R	02	PATRN_MOVAB	00000962	R	02
PATRN_CVTWH	00000A1A	R	02	PATRN_MOVAL	00000990	R	02
PATRN_CVTWL	00000890	R	02	PATRN_MOVAO	00000A3F	R	02
PATRN_DECB	00000951	R	02	PATRN_MOVAQ	00000937	R	02
PATRN_DECL	0000098E	R	02	PATRN_MOVAW	000008AC	R	02
PATRN_DECW	00000977	R	02	PATRN_MOVB	00000943	R	02
PATRN_DIVB2	0000093C	R	02	PATRN_MOVC3	00000878	R	02
PATRN_DIVB3	0000093F	R	02	PATRN_MOVC5	00000881	R	02
PATRN_DIVD2	000008ED	R	02	PATRN_MOVD	0000090B	R	02
PATRN_DIVD3	000008F0	R	02	PATRN_MOVF	000008CF	R	02
PATRN_DIVF2	000008B1	R	02	PATRN_MOVG	000009F2	R	02
PATRN_DIVF3	000008B4	R	02	PATRN_MOVH	00000A25	R	02
PATRN_DIVG2	000009D4	R	02	PATRN_MOVL	00000980	R	02
PATRN_DIVG3	000009D7	R	02	PATRN_MOVO	00000A3C	R	02
PATRN_DIVH2	00000A07	R	02	PATRN_MOVP	00000896	R	02
PATRN_DIVH3	00000A0A	R	02	PATRN_MOVPSL	0000098A	R	02
PATRN_DIVL2	00000979	R	02	PATRN_MOVQ	00000934	R	02
PATRN_DIVL3	0000097C	R	02	PATRN_MOVTC	00000887	R	02
PATRN_DIVP	0000086B	R	02	PATRN_MOVTUC	00000887	R	02
PATRN_DIVW2	000008E4	R	02	PATRN_MOVW	00000969	R	02
PATRN_DIVW3	00000965	R	02	PATRN_MOVZBL	00000953	R	02
PATRN_EDITPC	0000089E	R	02	PATRN_MOVZBW	00000956	R	02
PATRN_EDIV	0000092D	R	02	PATRN_MOVZWL	00000890	R	02
PATRN_EMODD	00000913	R	02	PATRN_MTPR	00000987	R	02
PATRN_EMODF	000008D7	R	02	PATRN_MULB2	0000093C	R	02
PATRN_EMODG	000009FA	R	02	PATRN_MULB3	0000093F	R	02
PATRN_EMODH	00000A2D	R	02	PATRN_MULD2	000008ED	R	02
PATRN_EMUL	00000928	R	02	PATRN_MULD3	000008F0	R	02
PATRN_EXTV	0000099C	R	02	PATRN_MULF2	000008B1	R	02
PATRN_EXTZV	0000099C	R	02	PATRN_MULF3	000008B4	R	02
PATRN_FFC	0000099C	R	02	PATRN_MULG2	000009D4	R	02
PATRN_FFS	0000099C	R	02	PATRN_MULG3	000009D7	R	02
PATRN_HALT	00000846	R	02	PATRN_MULH2	00000A07	R	02
PATRN_INCB	00000951	R	02	PATRN_MULH3	00000A0A	R	02
PATRN_INCL	0000098E	R	02	PATRN_MULL2	00000979	R	02
PATRN_INCW	00000977	R	02	PATRN_MULL3	0000097C	R	02
PATRN_INDEX	0000084C	R	02	PATRN_MULP	0000086B	R	02
PATRN_INSQHI	000008E7	R	02	PATRN_MULW2	000008E4	R	02

```
PATRN_MULW3     00000965 R      02      PATRN_XORL3             0000097C R      02
PATRN_NOP       00000846 R      02      PATRN_XORW2             000008E4 R      02
PATRN_POLYD     00000919 R      02      PATRN_XORW3             00000965 R      02
PATRN_POLYF     000008DD R      02      PSL                     = 00000054
PATRN_POLYG     00000A00 R      02      PSL$S_CURMOD            = 00000002
PATRN_POLYH     00000A33 R      02      PSL$V_CURMOD            = 00000018
PATRN_POPR      00000975 R      02      PSL$V_FPD               = 0000001B
PATRN_PROBER    00000858 R      02      PSL$V_TBIT              = 00000004
PATRN_PROBEW    00000858 R      02      PSL$V_TP                = 0000001E
PATRN_PUSHAB    00000864 R      02      READ_ADDRS              = FFFFFE78
PATRN_PUSHAL    00000993 R      02      READ_CHECK              00000DCE R      02
PATRN_PUSHAO    00000A42 R      02      READ_FAULT              00000EEB R      02
PATRN_PUSHAQ    0000093A R      02      READ_OPERANDS           = FFFFFEB8
PATRN_PUSHAW    000008AF R      02      READ_REASON             00000F26 R      02
PATRN_PUSHL     0000098C R      02      READ_REG                00000BC7 R      02
PATRN_PUSHR     00000975 R      02      REASON_HANDLER          00000F32 R      02
PATRN_REI       00000846 R      02      REGISTER_MODE           00000B80 R      02
PATRN_REMQHI    000008EA R      02      REG_AP                  = 00000044
PATRN_REMQTI    000008EA R      02      REG_DEF_MODE            00000C05 R      02
PATRN_REMQUE    0000085F R      02      REG_FP                  = 00000048
PATRN_RET       00000846 R      02      REG_PC                  = 00000050
PATRN_ROTL      00000959 R      02      REG_R0                  = 00000014
PATRN_RSB       00000846 R      02      REG_SP                  = 0000004C
PATRN_SBWC      00000979 R      02      RESIGNAL                00000FD7 R      02
PATRN_SCANC     0000087C R      02      ROPRAND_CHECK           00000DED R      02
PATRN_SKPC      000008A3 R      02      SAVE_ALIGN              = FFFFFFFF
PATRN_SOBGEQ    000009B4 R      02      SAVE_AP                 = 00000008
PATRN_SOBGTR    000009B4 R      02      SAVE_DEPTH              = FFFFFFF8
PATRN_SPANC     0000087C R      02      SAVE_MASK               = 00000006
PATRN_SUBB2     0000093C R      02      SAVE_PARCNT             = FFFFFFFE
PATRN_SUBB3     0000093F R      02      SAVE_PC                 = 00000010
PATRN_SUBD2     000008ED R      02      SAVE_SIGARGS            = FFFFFDDC
PATRN_SUBD3     000008F0 R      02      SF$L_SAVE_FP            = 0000000C
PATRN_SUBF2     000008B1 R      02      SHORT_LOCAL             = FFFFFFF8
PATRN_SUBF3     000008B4 R      02      SIGARGS                 = 00000004
PATRN_SUBG2     000009D4 R      02      SIGNAL                  000010DC R      02
PATRN_SUBG3     000009D7 R      02      SIGNAL_ARGS             = 0000000C
PATRN_SUBH2     00000A07 R      02      SIGNAL_START            0000106F R      02
PATRN_SUBH3     00000A0A R      02      SS$_ACCVIO              = 0000000C
PATRN_SUBL2     00000979 R      02      SS$_BREAK               = 00000414
PATRN_SUBL3     0000097C R      02      SS$_FLTDIV_F            = 000004BC
PATRN_SUBP4     00000866 R      02      SS$_FLTOVF_F            = 000004B4
PATRN_SUBP6     0000086B R      02      SS$_FLTUND_F            = 000004C4
PATRN_SUBW2     000008E4 R      02      SS$_OPCCUS              = 00000434
PATRN_SUBW3     00000965 R      02      SS$_OPCDEC              = 0000043C
PATRN_SVPCTX    00000846 R      02      SS$_RADRMOD             = 0000044C
PATRN_TSTB      0000094F R      02      SS$_RESIGNAL            = 00000918
PATRN_TSTD      00000911 R      02      SS$_ROPRAND             = 00000454
PATRN_TSTF      000008D5 R      02      SS$_TBIT                = 00C00464
PATRN_TSTG      000009F8 R      02      SS$_UNWIND              = 00000920
PATRN_TSTH      00000A2B R      02      STD_OPCODE              000000FF R      02
PATRN_TSTL      0000098C R      02      SYS$CALL_HANDL          ********   X    00
PATRN_TSTW      00000975 R      02      SYS$SRCHANDLER          ********   X    00
PATRN_XFC       00000846 R      02      SYS$UNWIND              ********   X    00
PATRN_XORB2     0000093C R      02      TAB_1BYTE               00000446 R      02
PATRN_XORB3     0000093F R      02      TAB_2BYTE               00000646 R      02
PATRN_XORL2     00000979 R      02      TEMP                    = 00000058
```

```
TEST_FRAME                      00000398 R      02
USER_ACTION                   = 0000000C
USER_ACT_ADR                  = FFFFFDE0
USER_ACT_ARG                  = FFFFFDE8
USER_ACT_ENV                  = FFFFFDE4
USER_ARG                      = 00000010
USER_SIGNAL                     00000FE8 R      02
V_REGISTER                    = 00000000
V_RESIGNAL                    = 00000001
WORD_DEF_MODE                   00000D1D R      02
WORD_DISP_MODE                  00000CF4 R      02
WRITE_ADDRS                   = FFFFFE38
WRITE_CHECK                     00000E33 R      02
WRITE_FAULT                     00000F5C R      02
WRITE_REASON                    00000F97 R      02
WRITE_REG                       00000BBA R      02
```

Psect synopsis

PSECT name	Allocation	PSECT No.	Attributes
. ABS .	00000000 (0.)	00 (0.)	NOPIC USR CON ABS LCL NOSHR NOEXE NORD NOWRT NOVEC BYTE
ABS	00000000 (0.)	01 (1.)	NOPIC USR CON ABS LCL NOSHR EXE RD WRT NOVEC BYTE
_LIB$CODE	000010E2 (4322.)	02 (2.)	PIC USR CON REL LCL SHR EXE RD NOWRT NOVEC LONG

Performance indicators

Phase	Page faults	CPU Time	Elapsed Time
Initialization	32	00:00:00.05	00:00:00.99
Command processing	128	00:00:00.31	00:00:03.29
Pass 1	422	00:00:15.30	00:01:47.95
Symbol table sort	0	00:00:01.23	00:00:05.07
Pass 2	414	00:00:04.98	00:00:18.11
Symbol table output	1	00:00:00.24	00:00:01.09
Psect synopsis output	1	00:00:00.01	00:00:00.01
Cross-reference output	0	00:00:00.00	00:00:00.00
Assembler run totals	1000	00:00:22.12	00:02:16.51

The working set limit was 2100 pages.
115302 bytes (226 pages) of virtual memory were used to buffer the intermediate code.
There were 70 pages of symbol table space allocated to hold 1132 non-local and 112 local symbols.
3138 source lines were read in Pass 1, producing 29 object records in Pass 2.
18 pages of virtual memory were used to define 14 macros.

Macro library statistics

Macro library name	Macros defined
_255DUA28:[SYSLIB]STARLET.MLB;2	9

771 GETS were required to define 9 macros.

There were no errors, warnings or information messages.

```
MACRO/ENABLE=SUPPRESSION/DISABLE=(GLOBAL,TRACEBACK)/LIS=LIS$:LIBDECODF/OBJ=OBJ$:LIBDECODF MSRC$:LIBDECODF/UPDATE=(ENH$:LIBDECODF)
```

0205 AH-BT13A-SE
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY