APPENDIX A: SAMPLE DEBUGGING DIALOGUE

This appendix contains a sample ALADDIN dialogue from debugging the program in Figure A1 which finds the maximum, minimum and total of a list of positive numbers.

```
      000100          .LOC  100
00100 030123 START:   LDA   2,.LIST ; ADDRESS OF LIST INTO AC2
00101 021000          LDA   0,0,2   ; FIRST NUMBER IN LIST INTO AC0
00102 040131          STA   0,MAX   ; FIRST NUMBER IS THE MAX,
00103 040132          STA   0,MIN   ;                      MIN,
00104 040133          STA   0,TOTAL ;                  AND TOTAL
00105 151400 NEXT:    INC   2,2     ; BUMP AC2 TO THE NEXT NUMBER
00106 021000          LDA   0,0,2   ; NEXT NUMBER INTO AC0
00107 101112          MOVL# 0,0,SZC ; CHECK FOR THE END OF THE LIST
00110 063077          HALT          ; YES -- STOP
00111 024131          LDA   1,MAX   ;
00112 106513          SUBL# 0,1,SNC ; CHECK IT AGAINST THE CURRENT MAX
00113 040131          STA   0,MAX   ; IT'S BIGGER, SO UPDATE THE MAX
00114 024132          LDA   1,MIN   ;
00115 122512          SUBL# 1,0,SZC ; CHECK IT AGAINST THE CURRENT MIN
00116 040132          STA   0,MIN   ; IT'S SMALLER, SO UPDATE THE MIN
00117 024133          LDA   1,TOTAL ;
00120 107000          ADD   0,1     ; ADD IT TO THE TOTAL
00121 044133          STA   1,TOTAL ;
00122 000105          JMP   NEXT    ; KEEP GOING
00123 000124 .LIST:   LIST          ;
00124 000005 LIST:    5             ; LIST OF NUMBERS
00125 000003          3             ;
00126 000002          2             ;
00127 000007          7             ;
00130 177777          -1            ; END-OF-LIST MARKER
      000001 MAX:     .BLK  1       ;
      000001 MIN:     .BLK  1       ;
      000001 TOTAL:   .BLK  1       ;
                      .END
```

Fig. A1. Sample program to be debugged.

```
? S
        SYMBOL TABLE

LOCATION      SYMBOL      ADDRESS
--------      ------      -------
 005372       *AC0         000213
 005376       *AC1         000214
 005402       *AC2         000215
 005406       *AC3         000216
 005412       *CBIT        000217
 005416       *PC          000220
 005422       START        000100
 005426       NEXT         000105
 005432       .LIST        000123
 005436       LIST         000124
 005442       MAX          000131
 005446       MIN          000132
 005452       TOTAL        000133

? B $ 1 = 1 $      (use an always-true assertion to run to completion)

? E
            ADDR      INST      CARRY     AC-0      AC-1      AC-2      AC-3
            ------    ------    -----     ------    ------    ------    ------
HALT:       000110    063077      1       177777    000021    000130    000177
PREVIOUS:   000107    101112      1       177777    000021    000130    000177

? I $ TOTAL = 21 $
   TRUE

? I $ MAX = 7 $
   FALSE          (test data used should have MAX = 7 -- error)

? I $ MIN = 2 $
   TRUE

? F
     DYNAMIC FLOW TRACE

  000107 ---> 000111 (00001)
  000122 ---> 000105 (00001)
  000107 ---> 000111 (00001)
  000122 ---> 000105 (00001)
  000107 ---> 000111 (00001)
  000112 ---> 000114 (00001)      (MAX update was skipped with the 7)
  000115 ---> 000117 (00001)
  000122 ---> 000105 (00001)

? B $ *PC <> 114 : *PC = 114 & MAX >= *AC0 $      (assert MAX update is
                                                   performed correctly)
? M 220/000102 100      (reset *PC to start of program)
```

```
? E
            ADDR     INST     CARRY    AC-0     AC-1     AC-2     AC-3
            ------   ------   -----    ------   ------   ------   ------
FAILURE:    000114   024132     1      000007   000002   000127   000177
PREVIOUS:   000112   106513     1      000007   000002   000127   000177

? M 112/106513 106512        (skip condition was wrong on MAX test -- should
                              be SZC -- i.e., skip if MAX - NUM >= 0)
? M 220/000114 100           (reset *PC)

? E
            ADDR     INST     CARRY    AC-0     AC-1     AC-2     AC-3
            ------   ------   -----    ------   ------   ------   ------
HALT:       000110   063077     1      177777   000021   000130   000177
PREVIOUS:   000107   101112     1      177777   000021   000130   000177

? I $ TOTAL = 21 & MIN = 2 & MAX = 7 $      (check all the results)
   TRUE                                     (all are correct)

?
```

APPENDIX B: SUMMARY OF ALADDIN COMMANDS

This appendix contains a summary of the ALADDIN command syntax.

Notes:

1) All user-supplied portions of the commands are underlined.

2) "n" is any valid accumulator number (0-3 for the NOVA).

3) "xxxxxx" and "yyyyyy" are any valid 16-bit octal strings (000000-177777).

4) "assertion text" can be any properly formed Boolean expression containing user program symbols, reserved ALADDIN symbols, octal constants, logical/relational operators and arithmetic operators; also, parenthesizing of subexpressions is allowed.

5) "aaaaaa" is any valid octal address (000000 to the maximum address of the machine being used).

Command	Meaning
A AC-n/xxxxxx (CR)	Examine contents (xxxxxx) of accumulator n and leave it unchanged.
A AC-n/xxxxxx yyyyyy (CR)	Examine contents (xxxxxx) of accumulator n and change it to yyyyyy.
B $assertion text$	Enter a breakpointing assertion.
E	Execute the user program subject to the latest breakpointing assertion.
F	Dump the dynamic flow table from the last execution.
I $assertion text$	Evaluate an immediate assertion.
M aaaaaa/xxxxxx (CR)	Examine contents (xxxxxx) of memory location aaaaaa and leave it unchanged.
M aaaaaa/xxxxxx yyyyyy (CR)	Examine contents (xxxxxx) of memory location aaaaaa and change it to yyyyyy.
S	Dump the debugger symbol table.

Fig. A2. ALADDIN command summary.

APPENDIX C: SUMMARY OF ALADDIN ERROR MESSAGES

Message	Meaning
...←INVALID ACCUMULATOR	Identifies an invalid accumulator number while processing an "A" command.
...←INVALID CHARACTER	Identifies an invalid character during input of an octal constant or address.
...←INVALID COMMAND	Identifies an invalid command mnemonic input to the command monitor.
*****ERROR***** ATXT OVERFLOW	Indicates overflow of the assertion text buffer while inputting an assertion.
*****ERROR***** INVALID TOKEN NEAR".."	Indicates the vicinity of an error found in the syntax of an assertion.
*****ERROR***** IPS OVERFLOW	Indicates overflow of the immediate polish stack while performing the syntax check of an assertion.
*****ERROR***** TOS OVERFLOW	Indicates overflow of the temporary operator stack while performing the syntax check of an assertion.
*****ERROR***** UNMATCHED PARENTHESES	Indicates an improperly parenthesized assertion.

Fig. A3. ALADDIN error message summary.

APPENDIX D: ALADDIN SOURCE PROGRAM LISTING

This appendix contains the complete source for the version of ALADDIN described in this thesis. This code was written in Data General's NOVA assembly language. It was assembled on a NOVA 1200 system with 32K of main memory and 3.75 megawords of disk storage running version 5.01 of the Real Time Disk Operating System (RDOS). The object code was executed on an 8K NOVA system and also a 4K SUPERNOVA system.

```
;*****************************************************************************
;*****************************************************************************
;       A L A D D I N   D E B U G G E R  --  V E R S I O N   1
;*****************************************************************************
;*****************************************************************************
;
;*****************************************************************************
;       P A G E   Z E R O   C O N S T A N T S   A N D   A D D R E S S E S
;*****************************************************************************
        .LOC    1
        TISR                    ; DIRECT INTERRUPTS TO "TISR", TELETYPE INPUT
                                ;                              SERVICE ROUTINE
        .LOC    140
AMASK:  077777                  ; 15-BIT ADDRESS MASK
AMPER:  "&                      ; ASCII "&"
ARROW:  137                     ; ASCII BACKWARDS ARROW
ASTRX:  "*                      ; ASCII "*"
ATXTM:  617                     ; ADDRESS OFFSET TO LAST ATXT ENTRY
ATXTP:  .BLK    1               ; ASSERTION TEXT BUFFER POINTER
BIT4:   004000                  ; BIT 4 MASK
BIT5:   002000                  ; BIT 5 MASK
BIT6:   001000                  ; BIT 6 MASK
BIT7:   000400                  ; BIT 7 MASK
BIT8:   000200                  ; BIT 8 MASK
BIT15:  000001                  ; BIT 15 MASK
CMASK:  000177                  ; ASCII CHARACTER MASK
COLON:  ":                      ; ASCII ":"
COMMA:  ",                      ; ASCII ","
CR:     015                     ; ASCII CARRIAGE RETURN
DFTMX:  11                      ; ADDRESS OFFSET TO LAST DFT VECTOR ENTRIES
DFTP:   .BLK    1               ; CURRENT NUMBER OF DFT VECTOR ENTRIES
DIG0:   "0                      ; ASCII "0"
DIG1:   "1                      ; ASCII "1"
DIG3:   "3                      ; ASCII "3"
DIG7:   "7                      ; ASCII "7"
```

```
DIG9:   "9                      ; ASCII "9"
DMASK:  000377                  ; MRI DISPLACEMENT FIELD MASK
DOT:    ".                      ; ASCII "."
DSIGN:  "$                      ; ASCII "$"
ESC:    033                     ; ASCII ESCAPE
EQ:     "=                      ; ASCII "="
GT:     ">                      ; ASCII ">"
HMASK:  063077                  ; HALT INSTRUCTION MASK
IMASK:  001400                  ; MRI INDEX FIELD MASK
IPSM:   77                      ; ADDRESS OFFSET TO LAST IPS ENTRY
IPSP:   .BLK    1               ; IMMEDIATE ASSERTION POLISH STACK POINTER
JMASK:  170000                  ; MRI JMP/JSR MASK
LETA:   "A                      ; ASCII "A"
LETZ:   "Z                      ; ASCII "Z"
LF:     012                     ; ASCII LINE FEED
LOGF:   000000                  ; LOGICAL FALSE IN POLISH STACK EVALUATION
LOGT:   000001                  ; LOGICAL TRUE IN POLISH STACK EVALUATION
LPARN:  "(                      ; ASCII "("
LT:     "<                      ; ASCII "<"
MSIGN:  "-                      ; ASCII "-"
NEG1:   -1                      ; -1
NEWA0:  .BLK    1               ; CURRENT USER AC0
NEWA1:  .BLK    1               ; CURRENT USER AC1
NEWA2:  .BLK    1               ; CURRENT USER AC2
NEWA3:  .BLK    1               ; CURRENT USER AC3
NEWCY:  .BLK    1               ; CURRENT USER CARRY BIT
NEWPC:  .BLK    1               ; CURRENT USER PROGRAM COUNTER
NMASK:  177400                  ; NEGATIVE DISPLACEMENT SIGN BIT EXTENSION MASK
OLDA0:  .BLK    1               ; PREVIOUS USER AC0
OLDA1:  .BLK    1               ; PREVIOUS USER AC1
OLDA2:  .BLK    1               ; PREVIOUS USER AC2
OLDA3:  .BLK    1               ; PREVIOUS USER AC3
OLDCY:  .BLK    1               ; PREVIOUS USER CARRY BIT
OLDPC:  .BLK    1               ; PREVIOUS USER PROGRAM COUNTER
OMASK:  000007                  ; OCTAL DIGIT BINARY VALUE MASK
```

```
PCINC: .BLK    1           ; PROGRAM COUNTER INCREMENTING FACTOR
PDUMP: .BLK    1           ; FLAG TO ENABLE DUMPING THE PREVIOUS CPU STATE
PMASK: 000000              ; POSITIVE DISPLACEMENT SIGN BIT EXTENSION MASK
POS1:  1                   ; +1
POS2:  2                   ; +2
POS3:  3                   ; +3
POS4:  4                   ; +4
POS5:  5                   ; +5
POS6:  6                   ; +6
POS7:  7                   ; +7
POS8:  10                  ; +8
POS9:  11                  ; +9
POS10: 12                  ; +10
POS11: 13                  ; +11
PSIGN: "+                  ; ASCII "+"
QUOTE: 042                 ; ASCII QUOTE
RPARN: ")                  ; ASCII ")"
SCOLN: ";                  ; ASCII ";"
SLASH: "/                  ; ASCII "/"
SPACE: 040                 ; ASCII " "
SPSP:  .BLK    1           ; SCRATCHPAD POLISH STACK POINTER
SYMNO: .BLK    1           ; CURRENT NUMBER OF ENTRIES IN USER SYMBOL TABLE
TTMSK: 177775              ; TELETYPE INPUT INTERRUPT ENABLE MASK
TOSM:  77                  ; ADDRESS OFFSET TO LAST TOS ENTRY
TOSP:  .BLK    1           ; TEMPORARY OPERATOR STACK POINTER
ZERO:  0                   ; 0
.ACM:  ACM                 ; ADDRESS OF "ACM", ACCUMULATOR EXAMINE/MODIFY HANDLER
.ALA:  ALA                 ; ADDRESS OF "ALA", ALADDIN MONITOR ROUTINE
.ALA1: ALA1                ; ADDRESS OF "ALA1", ENTRY POINT IN "ALA"
                           ;                    FOR COMMAND PROMPTING
.ALB:  ALB                 ; ADDRESS OF "ALB", ALPHABETIC CHARACTER VERIFIER
.ALN:  ALN                 ; ADDRESS OF "ALN", ALPHANUMERIC CHARACTER VERIFIER
.ATPT: ATPT                ; ADDRESS OF ATPT, ALLOWABLE TOKEN PAIR TABLE
.ATXT: ATXT                ; ADDRESS OF ATXT, ASSERTION TEXT BUFFER
.BCHK: BCHK                ; ADDRESS OF "BCHK", BREAKPOINTING ASSERTION
```

```
                        ;                          EVALUATION ROUTINE
.BCM:   BCM             ; ADDRESS OF "BCM", BREAKPOINTING ASSERTION HANDLER
.BPS:   BPS             ; ADDRESS OF BPS, BREAKPOINTING ASSERTION POLISH STACK
.BRK:   BRK             ; ADDRESS OF "BRK", BREAKPOINT ROUTINE
.B2D:   B2D             ; ADDRESS OF "B2D", BINARY TO DECIMAL CONVERSION ROUTINE
.B2O:   B2O             ; ADDRESS OF "B2O", BINARY TO OCTAL CONVERSION ROUTINE
.CBIT:  CBIT            ; ADDRESS OF "CBIT", CARRY BIT DUMPING ROUTINE
.CPY:   CPY             ; ADDRESS OF "CPY", CPU STATE VECTOR COPYING ROUTINE
.DFTC:  DFTC            ; ADDRESS OF DFTC, DFT COUNT VECTOR
.DFTD:  DFTD            ; ADDRESS OF DFTD, DFT DESTINATION VECTOR
.DFTS:  DFTS            ; ADDRESS OF DFTS, DFT SOURCE VECTOR
.DMP:   DMP             ; ADDRESS OF "DMP", CPU STATE DUMP ROUTINE
.ECAC:  ECAC            ; ADDRESS OF "ECAC", CURRENT ASSERTION TEXT
                        ;                          CHARACTER EXAMINING ROUTINE
.ECM:   ECM             ; ADDRESS OF "ECM", EXECUTION HANDLER
.ENAC:  ENAC            ; ADDRESS OF "ENAC", NEXT ASSERTION TEXT
                        ;                          CHARACTER EXAMINING ROUTINE
.ERR1:  ERR1            ; ADDRESS OF ERR1(INVALID CHARACTER) ERROR MESSAGE
.ERR2:  ERR2            ; ADDRESS OF ERR2(TOS OVERFLOW) ERROR MESSAGE
.ERR3:  ERR3            ; ADDRESS OF ERR3(IPS OVERFLOW) ERROR MESSAGE
.ERR4:  ERR4            ; ADDRESS OF ERR4(INVALID TOKEN) ERROR MESSAGE
.ERR5:  ERR5            ; ADDRESS OF ERR5(UNMATCHED PARENTHESES) ERROR MESSAGE
.ERR6:  ERR6            ; ADDRESS OF ERR6(ATXT OVERFLOW) ERROR MESSAGE
.ERR7:  ERR7            ; ADDRESS OF ERR7(INVALID ACCUMULATOR) ERROR MESSAGE
.ERR8:  ERR8            ; ADDRESS OF ERR8(INVALID COMMAND) ERROR MESSAGE
.FCM:   FCM             ; ADDRESS OF "FCM", DYNAMIC FLOW DUMP HANDLER
.GAT:   GAT             ; ADDRESS OF "GAT", ASSERTION TEXT INPUT ROUTINE
.GETC:  GETC            ; ADDRESS OF "GETC", TELETYPE INPUT(SINGLE CHARACTER)
.HLT:   HLT             ; ADDRESS OF "HLT", PROGRAM HALT ROUTINE
.ICM:   ICM             ; ADDRESS OF "ICM", IMMEDIATE ASSERTION HANDLER
.IPS:   IPS             ; ADDRESS OF IPS, IMMEDIATE ASSERTION POLISH STACK
.MCM:   MCM             ; ADDRESS OF "MCM", MEMORY EXAMINATION/MODIFICATION
                        ;                          HANDLER
.NACS:  NEWAO           ; ADDRESS OF CURRENT ACCUMULATOR SAVE AREA
.NTKN:  NTKN            ; ADDRESS OF "NTKN", TOKEN EXTRACTION ROUTINE
```

```
.OCT:  OCT      ; ADDRESS OF "OCT", OCTAL DIGIT VERIFICATION ROUTINE
.OUT1: OUT1     ; ADDRESSES
.OUT2: OUT2     ;
.OUT3: OUT3     ;
.OUT4: OUT4     ;           OF
.OUT5: OUT5     ;
.OUT6: OUT6     ;
.OUT7: OUT7     ;              OUTPUT
.OUT8: OUT8     ;
.OUT9: OUT9     ;
.OUTA: OUTA     ;                    TEXT
.OUTB: OUTB     ;
.OUTC: OUTC     ;
.OUTD: OUTD     ;                         BUFFERS
.OUTE: OUTE     ;
.OUTF: OUTF     ;
.OUTG: OUTG     ;
.O2B:  O2B      ; ADDRESS OF "O2B", OCTAL TO BINARY CONVERSION ROUTINE
.POPT: POPT     ; ADDRESS OF "POPT", TEMPORARY OPERAND STACK
                ;                    POP ROUTINE
.PSB:  PSB      ; ADDRESS OF "PSB", POLISH STACK BUILDING ROUTINE
.PSE:  PSE      ; ADDRESS OF "PSE", POLISH STACK EVALUATION ROUTINE
.PSHI: PSHI     ; ADDRESS OF "PSHI", IMMEDIATE POLISH STACK PUSH ROUTINE
.PSHT: PSHT     ; ADDRESS OF "PSHT", TEMPORARY OPERAND STACK
                ;                    PUSH ROUTINE
.PUTB: PUTB     ; ADDRESS OF "PUTB", TELETYPE OUTPUT(BUFFERED)
.PUTC: PUTC     ; ADDRESS OF "PUTC", TELETYPE OUTPUT(SINGLE CHARACTER)
.RNAC: RNAC     ; ADDRESS OF "RNAC", NEXT ASSERTION TEXT
                ;                    CHARACTER RETRIEVING ROUTINE
.SCM:  SCM      ; ADDRESS OF "SCM", SYMBOL TABLE DUMP HANDLER
.SPS:  SPS      ; ADDRESS OF SPS, SCRATCHPAD POLISH STACK
.SP3:  SP3      ; ADDRESS OF "SP3", TRIPLE SPACING ROUTINE
.SRCH: SRCH     ; ADDRESS OF "SRCH", SYMBOL TABLE SEARCH ROUTINE
.SYMR: SYMR     ; ADDRESS OF SYMR, RESERVED SYMBOL TABLE
.SYMU: SYMU     ; ADDRESS OF SYMU, USER SYMBOL TABLE
```

```
.TOS:  TOS                  ; ADDRESS OF TOS, TEMPORARY OPERAND STACK
.USER: .BLK   1             ; ADDRESS OF USER PROGRAM
;*****************************************************************************
;       P A G E   Z E R O   C O N S T A N T S   A N D   A D D R E S S E S
;*****************************************************************************
;       A L A   A L A   A L A   A L A   A L A   A L A   A L A   A L A   A L A
;*****************************************************************************
;
; ALADDIN MONITOR
;
;       ENTRY:      ALA
;       INPUT:      NONE
;       OUTPUT:     NONE
;       REQUIRED:   "ACM", "BCM", "ECM", "FCM", "GETC",
;                   "ICM", "MCM", "PUTB", "PUTC", "SCM"
;
        .LOC    400
ALA:    IORST               ; CLEAR ALL I/O DEVICES
        INTDS               ; DISABLE ALL INTERRUPTS
        STA     0,NEWA0     ; INITIALIZE CURRENT USER AC0
        STA     1,NEWA1     ; INITIALIZE CURRENT USER AC1
        STA     2,NEWA2     ; INITIALIZE CURRENT USER AC2
        STA     3,NEWA3     ; INITIALIZE CURRENT USER AC3
        MOVL    0,0         ; INITIALIZE CURRENT USER CARRY BIT
        LDA     1,BIT15     ;
        AND     1,0         ;
        STA     0,NEWCY     ;
        LDA     0,.USER     ; INITIALIZE CURRENT USER PROGRAM COUNTER
        STA     0,NEWPC     ;
ALA1:   LDA     2,.PUTB     ; OUTPUT THE COMMAND PROMPT TEXT
        LDA     0,.OUT1     ;
        STA     0,0,2       ;
        JSR     1,2         ;
        JSR     @.GETC      ; AC0 <-- THE COMMAND CHARACTER
```

```
        JSR     @.PUTC      ; ECHO IT
        LDA     2,.CLIS     ; AC2 <-- ADDRESS OF COMMAND LIST
ALA2:   LDA     1,0,2       ; AC1 <-- NEXT CHARACTER IN COMMAND LIST
        MOV#    1,1,SNR     ; CHECK FOR THE END OF THE LIST
        JMP     ALA3        ; YES -- INVALID COMMAND
        SUB#    0,1,SNR     ; NO -- CHECK IF THIS IS THE CHARACTER INPUT
        JMP     ALA4        ; YES -- BRANCH TO THE APPROPRIATE HANDLER
        INC     2,2         ; NO -- GO BACK AND TRY THE NEXT CHARACTER
                            ; IN THE COMMAND LIST
        JMP     ALA2        ;
ALA3:   LDA     2,.PUTB     ; FLAG THE COMMAND CHARACTER AS INVALID
        LDA     0,.ERR8     ;
        STA     0,0,2       ;
        JSR     1,2         ;
        JMP     ALA1        ;
ALA4:   LDA     1,.CLIS     ; CALCULATE THE DISPLACEMENT INTO THE JUMP TABLE --
                            ; .ALA5 + 2 * (ADDRESS IN CLIS OF CHARACTER - .CLIS) --
                            ; AND BRANCH TO THE APPROPRIATE COMMAND HANDLER
        SUB     1,2         ;
        MOVZL   2,1         ;
        LDA     2,.ALA5     ;
        ADD     1,2         ;
        JMP     0,2         ;
.ALA5:  ALA5                ; ADDRESS OF COMMAND HANDLER JUMP TABLE
ALA5:   JSR     @.ACM       ; "A" COMMAND -- ACCUMULATOR EXAMINE/MODIFY
        JMP     ALA1        ;
        JSR     @.BCM       ; "B" COMMAND -- BREAKPOINTING ASSERTION
        JMP     ALA1        ;
        JSR     @.ECM       ; "E" COMMAND -- EXECUTE
        JMP     ALA1        ;
        JSR     @.FCM       ; "F" COMMAND -- FLOW DUMP
        JMP     ALA1        ;
        JSR     @.ICM       ; "I" COMMAND -- IMMEDIATE ASSERTION
        JMP     ALA1        ;
        JSR     @.MCM       ; "M" COMMAND -- MEMORY EXAMINE/MODIFY
```

```
        JMP     ALA1        ;
        JSR     @.SCM       ; "S" COMMAND -- SYMBOL TABLE DUMP
        JMP     ALA1        ;
.CLTS:  CLIS                ; ADDRESS OF COMMAND LIST
CLTS:   "A                  ; COMMAND LIST
        "B
        "E
        "F
        "I
        "M
        "S
        0
;
;*************************************************************************
;       A L A   A L A   A L A   A L A   A L A   A L A   A L A   A L A   A L A
;*************************************************************************
;*************************************************************************
;       A C M   A C M   A C M   A C M   A C M   A C M   A C M   A C M   A C M
;*************************************************************************
;
; "ACM" HANDLES THE ACCUMULATOR EXAMINE/MODIFY COMMAND
;
;       ENTRY:      ACM
;       INPUT:      NONE
;       OUTPUT:     NONE
;       REQUIREL:   "B20", "GETC", "O2B", "PUTB", "PUTC"
;
ACM:    STA     3,SACM      ; SAVE RETURN ADDRESS
        LDA     2,.PUTB     ; OUTPUT THE ACCUMULATOR NUMBER PROMPT
        LDA     0,.OUTB     ;
        STA     0,0,2       ;
        JSR     1,2         ;
        JSR     @.GETC      ; AC0 <-- ACCUMULATOR NUMBER
        JSR     @.PUTC      ; ECHO IT
        LDA     1,DIG0      ;
```

```
LDA     2,D1G3      ;
ADCZ#   2,0,SNC     ; SKIP IF AC0(ACCUMULATOR) > AC2(ASCII "3")
ADCZ#   0,1,SZC     ; SKIP IF AC0(ACCUMULATOR) >= AC1(ASCII "0")
JMP     ACM1        ; INVALID ACCUMULATOR NUMBER
LDA     1,OMASK     ; VALID ACCUMULATOR NUMBER -- MASK OFF ALL BUT
                    ; BITS 13-15 TO GET ITS BINARY VALUE FOR OFFSETTING
                    ; INTO THE ACCUMULATOR SAVE AREA
AND     1,0         ;
LDA     2,.NACS     ;
ADD     0,2         ;
STA     2,AADDR     ; AC2 <-- ADDRESS OF ACCUMULATOR'S SAVE LOCATION
LDA     0,SLASH     ; OUTPUT A SLASH
JSR     @.PUTC      ;
LDA     0,@AADDR    ; OUTPUT THE ACCUMULATOR'S CURRENT CONTENTS
LDA     2,.B20      ;
STA     0,0,2       ;
JSR     1,2         ;
LDA     0,SPACE     ; OUTPUT THE UPDATING CONTENTS PROMPT
JSR     @.PUTC      ;
LDA     2,.02B      ; CALL "02B" TO LOOK FOR A STRING TO UPDATE
                    ; THE ACCUMULATOR'S CONTENTS
JSR     3,2         ;
LDA     2,.02B      ;
LDA     0,2,2       ;
LDA     1,CR        ;
SUB#    0,1,SZR     ; CHECK IF THE "02B" STRING DELIMITING CHARACTER
                    ; WAS A CARRIAGE RETURN
JMP     ACM2        ; NO -- INVALID INPUT
LDA     0,0,2       ;
MOV#    0,0,SNR     ; YES -- CHECK THE "02B" STRING VALIDITY FLAG
JMP     @SACM       ; NO STRING INPUT -- JUST A CARRIAGE RETURN
LDA     0,1,2       ; A VALID UPDATING STRING -- REPLACE THE ACCUMULATOR'S
                    ; CONTENTS AND RETURN
STA     0,@AADDR    ;
JMP     @SACM       ;
```

```
ACM1:   JSR     @.PUTC      ; FLAG THE INVALID ACCUMULATOR NUMBER AND RETURN
        LDA     2,.PUTB     ;
        LDA     0,.ERR7     ;
        STA     0,0,2       ;
        JSR     1,2         ;
        JMP     @SACM       ;
ACM2:   JSR     @.PUTC      ; FLAG THE INVALID INPUT CHARACTER AND RETURN
        LDA     2,.PUTB     ;
        LDA     0,.ERR1     ;
        STA     0,0,2       ;
        JSR     1,2         ;
        JMP     @SACM       ;
AACDR:  .BLK    1           ; ADDRESS OF ACCUMULATOR SAVE LOCATION BEING
                            ; EXAMINED/MODIFIED
SACM:   .BLK    1           ; "ACM" RETURN ADDRESS
;
;*****************************************************************************
;       A C M   A C M   A C M   A C M   A C M   A C M   A C M   A C M   A C M
;*****************************************************************************
;*****************************************************************************
;       B C M   B C M   B C M   B C M   B C M   B C M   B C M   B C M   B C M
;*****************************************************************************
;
; "BCM" HANDLES THE BREAKPOINT ASSERTION COMMAND
;
;       ENTRY:      BCM
;       INPUT:      NONE
;       OUTPUT:     NONE
;       REQUIRED:   "PSB"
;
BCM:    STA     3,SBCM      ; SAVE RETURN ADDRESS
        LDA     2,.PSB      ; CALL "PSB" TO INPUT AN ASSERTION AND BUILD
                            ; ITS POLISH STACK IN THE IPS STACK
        JSR     1,2         ;
        LDA     0,@.PSB     ;
```

```
        MOV#    0,0,SNR     ; CHECK FOR A VALID STACK
        JMP     @SBCM       ; NO -- JUST RETURN
        LDA     2,,IPS      ; YES -- COPY THE IPS STACK TO THE BPS STACK
        LDA     3,,BPS      ;
        LDA     0,0,2       ;
        STA     0,0,3       ;
        LDA     1,1,2       ;
        STA     1,1,3       ;
BCM1:   INC     2,2         ;
        INC     2,2         ;
        INC     3,3         ;
        INC     3,3         ;
        LDA     0,0,2       ;
        STA     0,0,3       ;
        LDA     1,1,2       ;
        STA     1,1,3       ;
        MOVL#   0,0,SNC     ; CHECK FOR A FENCE
        JMP     BCM1        ; NO -- MOVE THE NEXT ENTRY
        JMP     @SBCM       ; YES -- RETURN
SBCM:   .BLK    1           ; "BCM" RETURN ADDRESS
;
;*****************************************************************************
;       B C M   B C M   B C M   B C M   B C M   B C M   B C M   B C M   B C M
;*****************************************************************************
;*****************************************************************************
;       E C M   E C M   E C M   E C M   E C M   E C M   E C M   E C M   E C M
;*****************************************************************************
;
; "ECM" HANDLES THE EXECUTE COMMAND
;
;       ENTRY:      ECM
;       INPUT:      NONE
;       OUTPUT:     NONE
;       REQUIRED:   "BCHK", "BRK", "CPY", "HLT"
;
```

```
ECM:    STA     3,SECM      ; SAVE RETURN ADDRESS
        LDA     0,ZERO      ;
        STA     0,PDUMP     ; DISABLE THE PREVIOUS CPU STATE DUMP(PDUMP <-- 0)
        LDA     1,DFTMX     ;
        STA     1,DFTNO     ; INITIALIZE THE DFT POINTER TO THE LAST ENTRY OF THE
                            ; CIRCULAR BUFFERS(DFTNO <-- DFTMX) AND SET THE SOURCE
                            ; AND DESTINATION FIELDS OF THIS ENTRY BOTH TO ZERO
                            ; TO FORCE THE FIRST NON-SEQUENTIAL FLOW TO BE ENTERED
                            ; AT THE BEGINNING OF THE BUFFERS
        LDA     2,.DFTS     ;
        ADD     1,2         ;
        STA     0,0,2       ; (.DFTS + DFTMX) <-- 0
        LDA     2,.DFTD     ;
        ADD     1,2         ;
        STA     0,0,2       ; (.DFTD + DFTMX) <-- 0
        INC     1,1         ;
        STA     1,DFMX1     ; DFMX1 <-- DFTMX + 1
        LDA     2,.DFTC     ;
ECM1:   STA     0,0,2       ; ALSO, ZERO OUT THE ENTIRE DFT COUNT VECTOR
        INC     2,2         ;
        DSZ     DFMX1       ;
        JMP     ECM1        ;
        LDA     0,TIMSK     ; AC0 <-- 177775(TELETYPE INPUT INTERRUPT ENABLE MASK)
        MSKO    0           ; DISABLE INTERRUPTS FROM ALL DEVICES EXCEPT
                            ; THE TELETYPE INPUT
        NIOS    TTI         ; START THE TELETYPE INPUT
ECM2:   LDA     2,.BCHK     ; CALL "BCHK" TO EVALUATE THE BREAKPOINT ASSERTION
        JSR     1,2         ;
        LDA     0,@.BCHK    ;
        MOV#    0,0,SZR     ; CHECK IF IT EVALUATED TRUE OR FALSE
        JMP     ECM3        ; TRUE -- EXECUTE THE NEXT INSTRUCTION
        JSR     @.BRK       ; FALSE -- CALL "BRK" TO BREAKPOINT AND RETURN
        JMP     @SECM       ;
;
;       BEGIN THE INSTRUCTION INTERPRETATION PHASE
```

```
;
ECM3:   INTEN                   ; ALLOW THE TELETYPE INPUT TO INTERRUPT "ECM" ONLY
                                ; BEFORE INTERPRETING THE NEXT INSTRUCTION
        JMP     .+1             ;
        JMP     .+1             ;
        INTDS                   ;
        LDA     0,@NEWPC        ; AC0 <-- NEXT INSTRUCTION
        LDA     1,HMASK         ; AC1 <-- 063077(HALT MASK)
        SUB#    0,1,SZR         ; CHECK FOR A HALT INSTRUCTION
        JMP     ECM4            ; NOT A HALT -- CHECK FOR A JMP OR JSR
        JSR     @.HLT           ; YES -- CALL "HLT" TO HALT AND RETURN
        JMP     @SECM           ;
ECM4:   LDA     1,JMASK         ; AC1 <-- 170000(JUMP MASK)
        AND#    0,1,SNR         ; CHECK FOR A JMP OR JSR(BITS 0-3 ALL ZERO)
        JMP     ECM6            ; A JMP OR JSR
        LDA     1,POS3          ; NOT A JMP OR JSR, SO A SKIP OF ONE INSTRUCTION
                                ; IS THE ONLY POSSIBLE NON-SEQUENTIAL FLOW
        STA     1,PCINC         ; PCINC <-- 3
        STA     0,ECM5          ; TRANSFER THE INSTRUCTION "IN-LINE"
                                ; AND EXECUTE IT "PSEUDO-DIRECTLY" -- CALL "CPY"
                                ; TO COPY THE CURRENT CPU STATE SAVE AREA TO THE
                                ; PREVIOUS CPU STATE SAVE AREA, THEN RESTORE THE
                                ; CURRENT CPU STATE BEFORE ACTUALLY EXECUTING
                                ; THE INSTRUCTION
        JSR     @.CPY           ;
        LDA     0,NEWCY         ;
        MOVR    0,0             ;
        LDA     0,NEWA0         ;
        LDA     1,NEWA1         ;
        LDA     2,NEWA2         ;
        LDA     3,NEWA3         ;
ECM5:   .BLK    1               ; INSERTION POINT FOR INSTRUCTION
        DSZ     PCINC           ; IF THERE IS NO SKIP, PCINC WILL BE DECREMENTED
                                ; TWICE TO 1 -- IF THERE IS A SKIP, PCINC WILL BE
                                ; DECREMENTED ONCE TO 2
```

```
        DSZ     PCINC       ;
        STA     3,NEWA3     ; SAVE NEW CPU STATE
        STA     2,NEWA2     ;
        STA     1,NEWA1     ;
        STA     0,NEWA0     ;
        MOVL    0,0         ;
        LDA     1,BIT15     ;
        AND     1,0         ;
        STA     0,NEWCY     ;
        LDA     0,OLDPC     ; NEWPC <-- OLDPC + PCINC
        LDA     1,PCINC     ;
        ADD     1,0         ;
        LDA     1,AMASK     ;
        AND     0,1         ;
        STA     1,NEWPC     ;
        JMP     DFT1        ; THAT'S ALL -- HANDLE THE DFT UPDATE
ECM6:   JSR     @,CPY       ; A JMP OR JSR CAN ONLY MODIFY THE PROGRAM COUNTER
                            ; AND AC3, SO COPY THE CURRENT CPU STATE TO THE
                            ; PREVIOUS CPU STATE SAVE AREA, DETERMINE THE
                            ; NEW VALUE OF AC3(IF A JSR), AND DETERMINE THE
                            ; ADDRESSING SCHEME TO GET THE NEW PROGRAM COUNTER VALUE
        LDA     0,@NEWPC    ; AC0 <-- INSTRUCTION BEING EXECUTED
        LDA     1,BIT4      ; AC1 <-- 004000(JSR MASK)
        AND#    0,1,SNR     ; CHECK FOR A JSR
        JMP     ECM7        ; A JMP
        LDA     2,OLDPC     ; A JSR -- NEWA3 <-- OLDPC + 1
        INC     2,2         ;
        LDA     1,AMASK     ;
        AND     1,2         ;
        STA     2,NEWA3     ;
ECM7:   LDA     1,IMASK     ; AC1 <-- 001400(INDEX FIELD MASK)
        AND#    0,1,SZR     ; CHECK FOR PAGE ZERO ADDRESSING
        JMP     ECM8        ; NOT PAGE ZERO ADDRESSING
        LDA     2,DMASK     ; PAGE ZERO ADDRESSING,
                            ; SO EFFECTIVE ADDRESS <-- DISPLACEMENT FIELD
```

```
        AND     0,2             ;
        JMP     ECMB            ;
ECM8:   LDA     1,DMASK         ; AC1 <-- 000377(DISPLACEMENT FIELD MASK)
        AND     0,1             ; AC1 <-- DISPLACEMENT FIELD -- SINCE THE ADDRESSING
                                ; SCHEME IS NOT PAGE ZERO, THE DISPLACEMENT IS A
                                ; SIGNED 7-BIT NUMBER
        LDA     3,PMASK         ; ASSUME THE DISPLACEMENT IS POSITIVE(BIT 8 = 0) --
                                ; AC3 <-- 000000(POSITIVE SIGN BIT EXTENSION MASK)
        LDA     2,BIT8          ; AC2 <-- 000200(DISPLACEMENT SIGN BIT MASK)
        AND#    1,2,SZR         ; CHECK IF THE DISPLACEMENT IS POSITIVE AS ASSUMED
        LDA     3,NMASK         ; NO -- AC3 <-- 177400(NEGATIVE SIGN BIT EXTENSION MASK)
        ADD     3,1             ; AC1 <-- PROPERLY SIGNED 16-BIT DISPLACEMENT
        LDA     2,IMASK         ; AC2 <-- 001400(INDEX FIELD MASK)
        AND     0,2             ; AC2 <-- INDEX FIELD
        LDA     3,BIT7          ; AC3 <-- 000400(RELATIVE ADDRESSING INDEX MASK)
        SUB#    3,2,SZR         ; CHECK FOR RELATIVE ADDRESSING
        JMP     ECM9            ; NOT RELATIVE ADDRESSING
        LDA     2,OLDPC         ; RELATIVE ADDRESSING,
                                ; SO EFFECTIVE ADDRESS <-- OLDPC + DISPLACEMENT
        ADD     1,2             ;
        JMP     ECMB            ;
ECM9:   LDA     3,BIT6          ; AC3 <-- 001000(AC2-BASE ADDRESSING INDEX MASK)
        SUB#    3,2,SZR         ; CHECK FOR AC2-BASE ADDRESSING
        JMP     ECMA            ; NOT AC2-BASE ADDRESSING
        LDA     2,OLDA2         ; AC2-BASE ADDRESSING,
                                ; SO EFFECTIVE ADDRESS <-- OLDA2 + DISPLACEMENT
        ADD     1,2             ;
        JMP     ECMB            ;
ECMA:   LDA     2,OLDA3         ; MUST BE AC3-BASE ADDRESSING,
                                ; SO EFFECTIVE ADDRESS <-- OLDA3 + DISPLACEMENT
        ADD     1,2             ;
ECMB:   LDA     1,AMASK         ; EFFECTIVE ADDRESS IN AC2
        AND     1,2             ;
        LDA     0,@NEWPC        ; AC0 <-- INSTRUCTION BEING EXECUTED
        LDA     1,BIT5          ; AC1 <-- 002000(INDIRECT ADDRESSING MASK)
```

```
        AND#    0,1,SZR     ; CHECK FOR INDIRECT ADDRESSING
        LDA     2,0,2       ; INDIRECT -- EFFECTIVE ADDRESS <-- (EFFECTIVE ADDRESS)
        LDA     1,AMASK     ;
        AND     1,2         ;
        STA     2,NEWPC     ;
        JMP     DFT1        ; THAT'S ALL -- HANDLE THE DFT UPDATE
DFTX1:  .BLK    1           ; DFT VECTOR LOOP COUNTER
SECM:   .BLK    1           ; "ECM" RETURN ADDRESS
;
;       EXECUTION PHASE COMPLETE -- UPDATE THE DYNAMIC FLOW TRACE
;
DFT1:   LDA     0,POS1      ; AT LEAST ONE INSTRUCTION WAS EXECUTED, SO
                            ; ENABLE THE PREVIOUS CPU STATE DUMP(PDUMP <-- 1)
        STA     0,PDUMP     ;
        LDA     0,OLDPC     ; AC0 <-- OLDPC
        LDA     1,NEWPC     ; AC1 <-- NEWPC
        INC     0,2         ; AC2 <-- OLDPC + 1
        SUB#    1,2,SNR     ; COMPARE OLDPC AND NEWPC
        JMP     @.ECM2      ; JUST A SEQUENTIAL FLOW -- THROUGH WITH DFT UPDATE
        LDA     2,DFTNO     ; NON-SEQUENTIAL -- SEE IF IT'S THE LAST ENTRY MADE
        LDA     3,.DFTS     ;
        ADD     2,3         ;
        LDA     2,0,3       ; AC2 <-- LAST ENTRY'S SOURCE
        SUB#    0,2,SZR     ; CHECK IT AGAINST OLDPC
        JMP     DFT2        ; NO MATCH -- GO ADD THIS PAIR
        LDA     3,.DFTD     ;
        LDA     2,DFTNO     ;
        ADD     2,3         ;
        LDA     2,0,3       ; AC2 <-- LAST ENTRY'S DESTINATION
        SUB#    1,2,SZR     ; CHECK IT AGAINST NEWPC
        JMP     DFT2        ; NO MATCH -- GO ADD THIS PAIR
        LDA     2,DFTNO     ; BOTH SOURCE AND DESTINATION MATCH, SO INCREMENT
                            ; THE LAST ENTRY'S COUNT
        LDA     3,.DFTC     ;
        ADD     2,3         ; AC3 <-- ADDRESS OF LAST ENTRY'S COUNT FIELD
```

```
        ISZ     0,3         ; INCREMENT COUNT OF LAST DFTC ENTRY
        JMP     @.ECM2      ; THROUGH WITH DFT UPDATE
DFT2:   LDA     2,DFTNO     ; ADD THE (OLDPC,NEWPC,1) TRIPLE TO THE DFT VECTORS
                            ; AS THE NEXT ENTRY(DFTNO <-- DFTNO + 1)
        INC     2,2         ;
        LDA     3,DFTMX     ; AC3 <-- DFTMX(MAXIMUM VALUE OF DFTNO)
        ADCZ#   3,2,SZC     ; SKIP IF AC2(DFTNO + 1) <= AC3(DFTMX)
        LDA     2,ZERO      ; OVERFLOW -- CIRCLE AROUND TO THE FRONT OF THE DFT
                            ; VECTORS(DFTNO <-- 0)
        STA     2,DFTNO     ; DFTNO <-- (DFTNO + 1) MOD (SIZE OF DFT VECTORS)
        LDA     3,.DFTS     ;
        ADD     2,3         ;
        STA     0,0,3       ; (.DFTS + DFTNO) <-- OLDPC
        LDA     3,.DFTD     ;
        ADD     2,3         ;
        STA     1,0,3       ; (.DFTD + DFTNO) <-- NEWPC
        LDA     3,.DFTC     ;
        ADD     2,3         ;
        LDA     0,POS1      ;
        STA     0,0,3       ; (.DFTC + DFTNO) <-- 1
        JMP     @.ECM2      ; THROUGH WITH DFT UPDATE
.ECM2:  ECM2                ; ADDRESS OF "ECM2", ENTRY POINT IN "ECM" FOR
                            ; BREAKPOINT ASSERTION EVALUATION
;
;*****************************************************************************
;       E C M   E C M   E C M   E C M   E C M   E C M   E C M   E C M   F C M
;*****************************************************************************
;*****************************************************************************
;       F C M   F C M   F C M   F C M   F C M   F C M   F C M   F C M   F C M
;*****************************************************************************
;
; "FCM" HANDLES THE FLOW TRACE DUMP COMMAND
;
;       ENTRY:      FCM
;       INPUT:      NONE
```

```
;         OUTPUT:      NONE
;         REQUIRED:    "B2D", "B20", "PUTB"
;
FCM:    STA     3,SFCM      ; SAVE RETURN ADDRESS
        LDA     0,@.DFTC    ;
        MOV#    0,0,SNR     ; CHECK COUNT FIELD OF FIRST DFT ENTRY
        JMP     @SFCM       ; ZERO COUNT -- NO DFT ENTRIES
        LDA     2,.PUTB     ; OUTPUT DFT DUMP HEADING
        LDA     0,.OUT2     ;
        STA     0,0,2       ;
        JSR     1,2         ;
        LDA     0,DFTMX     ;
        INC     0,0         ;
        STA     0,DFMX2     ; DFMX2 <-- SIZE OF DFT VECTORS(DFTMX + 1)
FCM1:   LDA     0,DFTNO     ;
        INC     0,0         ; LOOK AT THE NEXT DFT ENTRY(DFTNO <-- DFTNO + 1)
        LDA     1,DFTMX     ; AC1 <-- DFTMX(MAXIMUM VALUE OF DFTNO)
        ADCZ#   1,0,SZC     ; SKIP IF AC0(DFTNO + 1) <= AC1(DFTMX)
        LDA     0,ZERO      ; CIRCLE AROUND TO THE FRONT OF THE DFT
                            ; VECTORS(DFTNO <-- 0)
        STA     0,DFTNO     ; DFTNO <-- (DFTNO + 1) MOD (SIZE OF DFT VECTORS)
        LDA     2,.DFTC     ;
        ADD     0,2         ;
        LDA     0,0,2       ;
        MOV#    0,0,SNR     ; CHECK THIS ENTRY'S COUNT FIELD
        JMP     FCM2        ; ZERO COUNT -- SKIP IT
        LDA     2,.PUTB     ;
        LDA     0,.OUT3     ;
        STA     0,0,2       ;
        JSR     1,2         ;
        LDA     2,.DFTS     ;
        LDA     1,DFTNO     ;
        ADD     1,2         ;
        LDA     0,0,2       ;
        LDA     2,.B20      ; OUTPUT THE SOURCE ADDRESS
```

```
        STA     0,0,2       ;
        JSR     1,2         ;
        LDA     2,.PUTB     ;
        LDA     0,.OUT4     ;
        STA     0,0,2       ;
        JSR     1,2         ;
        LDA     2,.DFTD     ;
        LDA     1,DFTNO     ;
        ADD     1,2         ;
        LDA     0,0,2       ;
        LDA     2,.B2D      ; OUTPUT THE DESTINATION ADDRESS
        STA     0,0,2       ;
        JSR     1,2         ;
        LDA     2,.PUTB     ;
        LDA     0,.OUT5     ;
        STA     0,0,2       ;
        JSR     1,2         ;
        LDA     2,.DFTC     ;
        LDA     1,DFTNO     ;
        ADD     1,2         ;
        LDA     0,0,2       ;
        LDA     2,.B2D      ; OUTPUT THE EXECUTION COUNT
        STA     0,0,2       ;
        JSR     1,2         ;
        LDA     2,.PUTB     ;
        LDA     0,.OUT6     ;
        STA     0,0,2       ;
        JSR     1,2         ;
FCM2:   DSZ     DFMX2       ; CHECK IF THAT'S A FULL PASS AROUND THE VECTORS YET
        JMP     FCM1        ; NO -- CONTINUE
        JMP     @SFCM       ; YES -- RETURN
DFMX2:  .BLK    1           ; DFT VECTORS LOOP COUNTER
SFCM:   .BLK    1           ; "FCM" RETURN ADDRESS
;
;*****************************************************************************
```

```
;        F C M   F C M   F C M   F C M   F C M   F C M   F C M   F C M   F C M
;*****************************************************************************
;*****************************************************************************
;        I C M   I C M   I C M   I C M   I C M   I C M   I C M   I C M   I C M
;*****************************************************************************
;
; "ICM" HANDLES THE IMMEDIATE ASSERTION COMMAND
;
;        ENTRY:       ICM
;        INPUT:       NONE
;        OUTPUT:      NONE
;        REQUIRED:    "PSB", "PSE", "PUTB"
;
ICM:    STA     3,SICM      ; SAVE RETURN ADDRESS
        LDA     2,.PSB      ; CALL "PSB" TO INPUT AN ASSERTION AND BUILD
                            ; ITS POLISH STACK IN THE IPS STACK
        JSR     1,2         ;
        LDA     0,@.PSB     ;
        MOV#    0,0,SNR     ; CHECK FOR A VALID STACK
        JMP     @SICM       ; NO -- JUST RETURN
        LDA     2,.PSE      ; YES -- CALL "PSE" TO EVALUATE IT
        LDA     0,.IPS      ;
        STA     0,0,2       ;
        JSR     2,2         ;
        LDA     2,.PSE      ;
        LDA     0,1,2       ;
        LDA     1,.OUTC     ; ASSUME IT WAS FALSE
        MOV#    0,0,SZR     ; CHECK THE "PSE" EVALUATION FLAG
        LDA     1,.OUTD     ; NO -- IT WAS TRUE
        LDA     2,.PUTB     ; OUTPUT THE APPROPRIATE VALUE MESSAGE AND RETURN
        STA     1,0,2       ;
        JSR     1,2         ;
        JMP     @SICM       ;
SICM:   .BLK    1           ; "ICM" RETURN ADDRESS
;
```

```
;*****************************************************************************
;        I C M   I C M   I C M   I C M   I C M   I C M   I C M   I C M   I C M
;*****************************************************************************
;        M C M   M C M   M C M   M C M   M C M   M C M   M C M   M C M   M C M
;*****************************************************************************
;
; "MCM" HANDLES THE MEMORY EXAMINE/MODIFY COMMAND
;
;        ENTRY:      MCM
;        INPUT:      NONE
;        OUTPUT:     NONE
;        REQUIRED:   "B20", "02B", "PUTB", "PUTC"
;
MCM:    STA     3,SMCM      ; SAVE RETURN ADDRESS
        LDA     0,SPACE     ; OUTPUT THE MEMORY ADDRESS PROMPT
        JSR     @.PUTC      ;
        LDA     2,.02B      ; CALL "02B" TO LOOK FOR AN OCTAL ADDRESS
        JSR     3,2         ;
        LDA     2,.02B      ;
        LDA     0,2,2       ;
        LDA     1,SLASH     ;
        SUB#    0,1,SZR     ; CHECK IF THE "02B" STRING DELIMITING CHARACTER
                            ; WAS A SLASH
        JMP     MCM1        ; NO -- INVALID INPUT
        LDA     1,0,2       ;
        MOV#    1,1,SNR     ; YES -- CHECK THE "02B" STRING VALIDITY FLAG
        JMP     MCM1        ; NO ADDRESS INPUT -- JUST A "/"
        JSR     @.PUTC      ; A VALID ADDRESS STRING -- DISPLAY ITS CONTENTS
                            ; AFTER THE SLASH
        LDA     0,1,2       ;
        LDA     1,AMASK     ;
        AND     1,0         ;
        STA     0,MADDR     ; MADDR <-- THE ADDRESS INPUT
        LDA     0,@MADDR    ; FETCH AND DISPLAY THIS LOCATION'S CONTENTS.
```

```
                                ; LEAVING IT OPEN FOR SUBSEQUENT MODIFICATION
        LDA     2,.B20          ;
        STA     0,0,2           ;
        JSR     1,2             ;
        LDA     0,SPACE         ; OUTPUT THE UPDATING CONTENTS PROMPT
        JSR     @.PUTC          ;
        LDA     2,.02B          ; CALL "02B" TO LOOK FOR A STRING TO UPDATE
                                ; THE LOCATION'S CONTENTS
        JSR     3,2             ;
        LDA     2,.02B          ;
        LDA     0,2,2           ;
        LDA     1,CR            ;
        SUB#    0,1,SZR         ; CHECK IF THE "02B" STRING DELIMITING
                                ; CHARACTER WAS A CARRIAGE RETURN
        JMP     MCM1            ; NO -- INVALID INPUT
        LDA     0,0,2           ;
        MOV#    0,0,SNR         ; YES -- CHECK THE "02B" STRING VALIDITY FLAG
        JMP     @SMCM           ; NO STRING INPUT -- JUST A CARRIAGE RETURN
        LDA     0,1,2           ; A VALID UPDATING STRING -- CHANGE THE LOCATION'S
                                ; CONTENTS AND RETURN
        STA     0,@MADDR        ;
        JMP     @SMCM           ;
MCM1:   JSR     @.PUTC          ; FLAG THE INVALID INPUT CHARACTER AND RETURN
        LDA     2,.PUTB         ;
        LDA     0,.ERR1         ;
        STA     0,0,2           ;
        JSR     1,2             ;
        JMP     @SMCM           ;
MADDR:  .BLK    1               ; ADDRESS OF LOCATION BEING EXAMINED/MODIFIED
SMCM:   .BLK    1               ; "MCM" RETURN ADDRESS
;
;*****************************************************************************
;       M C M   M C M   M C M   M C M   M C M   M C M   M C M   M C M   M C M
;*****************************************************************************
;*****************************************************************************
```

```
;        S C M   S C M   S C M   S C M   S C M   S C M   S C M   S C M   S C M
;*****************************************************************************
;
; "SCM" HANDLES THE SYMBOL TABLE DUMP COMMAND
;
;         ENTRY:      SCM
;         INPUT:      NONE
;         OUTPUT:     NONE
;         REQUIRED:   "B20", "PUTB", "PUTC", "SP3"
;
SCM:    STA     3,SSCM      ; SAVE RETURN ADDRESS
        LDA     2,.PUTB     ; OUTPUT SYMBOL TABLE DUMP HEADING
        LDA     0,.OUTE     ;
        STA     0,0,2       ;
        JSR     1,2         ;
        LDA     1,SYMNO     ; NSDMP <-- SYMNO(NUMBER OF USER SYMBOLS) +
                            ; 6(NUMBER OF RESERVED SYMBOLS)
        LDA     0,POS6      ;
        ADD     0,1         ;
        STA     1,NSDMP     ;
        LDA     0,.SYMR     ; SADDR <-- .SYMR(BEGINNING OF SYMBOL TABLE)
        STA     0,SADDR     ;
SCM1:   LDA     2,.PUTB     ; OUTPUT NEXT SYMBOL'S SYMBOL TABLE LOCATION
        LDA     0,.OUTF     ;
        STA     0,0,2       ;
        JSR     1,2         ;
        LDA     0,SADDR     ;
        LDA     2,.B20      ;
        STA     0,0,2       ;
        JSR     1,2         ;
        JSR     @.SP3       ;
        JSR     @.SP3       ;
        LDA     2,SADDR     ; OUTPUT SYMBOL
        LDA     0,0,2       ;
        MOVS    0,0         ;
```

```
LDA     1,CMASK     ;
AND     1,0,SNR     ;
LDA     0,SPACE     ;
JSR     @.PUTC      ; FIRST CHARACTER
LDA     0,0,2       ;
LDA     1,CMASK     ;
AND     1,0,SNR     ;
LDA     0,SPACE     ;
JSR     @.PUTC      ; SECOND CHARACTER
LDA     0,1,2       ;
MOVS    0,0         ;
LDA     1,CMASK     ;
AND     1,0,SNR     ;
LDA     0,SPACE     ;
JSR     @.PUTC      ; THIRD CHARACTER
LDA     0,1,2       ;
LDA     1,CMASK     ;
AND     1,0,SNR     ;
LDA     0,SPACE     ;
JSR     @.PUTC      ; FOURTH CHARACTER
LDA     0,2,2       ;
MOVS    0,0         ;
LDA     1,CMASK     ;
AND     1,0,SNR     ;
LDA     0,SPACE     ;
JSR     @.PUTC      ; FIFTH CHARACTER
JSR     @.SP3       ;
JSR     @.SP3       ;
LDA     0,SPACE     ;
JSR     @.PUTC      ;
LDA     2,SADDR     ; OUTPUT SYMBOL'S ADDRESS
LDA     0,3,2       ;
LDA     2,.B20      ;
STA     0,0,2       ;
JSR     1,2         ;
```

```
        LDA     0,POS4      ; SADDR <-- SADDR + 4 TO LOOK AT THE NEXT SYMBOL
                            ; TABLE ENTRY
        LDA     1,SADDR     ;
        ADD     0,1         ;
        STA     1,SADDR     ;
        DSZ     NSCMP       ; CHECK FOR THE END OF THE TABLE
        JMP     SCM1        ; NO -- DUMP THE NEXT SYMBOL
        JMP     @SSCM       ; YES -- RETURN
NSCMP:  .BLK    1           ; NUMBER OF SYMBOLS TO BE DUMPED
SADDR:  .BLK    1           ; ADDRESS OF SYMBOL TABLE ENTRY OF CURRENT SYMBOL
SSCM:   .BLK    1           ; "SCM" RETURN ADDRESS
;*****************************************************************************
;       S C M   S C M   S C M   S C M   S C M   S C M   S C M   S C M   S C M
;*****************************************************************************
;*****************************************************************************
;       A L B   A L B   A L B   A L B   A L B   A L B   A L B   A L B   A L B
;*****************************************************************************
;
; "ALB" PERFORMS ALPHABETIC("A" - "Z" AND ".") CHARACTER VERIFICATION
;
;       ENTRY:      ALB + 1
;       INPUT:      CHARACTER TO BE VERIFIED, IN ALB
;       OUTPUT:     VERIFICATION FLAG(0=INVALID,1=VALID), IN ALB
;       REQUIRED:   NONE
;
ALB:    .BLK    1           ; ON ENTRY, CHARACTER TO BE VERIFIED
                            ; ON EXIT, VERIFICATION FLAG(0=INVALID,1=VALID)
        STA     3,SALB      ; SAVE RETURN ADDRESS
        LDA     0,ALB       ; AC0 <-- CHARACTER TO BE VERIFIED
        LDA     1,ZERO      ; ASSUME THE CHARACTER IS INVALID(ALB <-- 0)
        STA     1,ALB       ;
        LDA     1,DOT       ; AC1 <-- ASCII "."
        SUB#    0,1,SNR     ; CHECK IF THE CHARACTER IS A "."
        JMP     ALB1        ; YES -- MARK THE CHARACTER AS VALID AND RETURN
        LDA     1,LETA      ; NO -- CHECK IF THE CHARACTER IS "A" - "Z"
```

```
        LDA     2,LETZ      ;
        ADCZ#   2,0,SNC     ; SKIP IF AC0(CHARACTER) > AC2(ASCII "Z")
        ADCZ#   0,1,SZC     ; SKIP IF AC0(CHARACTER) >= AC1(ASCII "A")
        JMP     @SALB       ; NOT "A" - "Z" EITHER, SO IT'S INVALID AS ASSUMED
ALB1:   LDA     0,POS1      ; MARK THE CHARACTER AS VALID(ALB <-- 1)
                            ; AND RETURN
        STA     0,ALB       ;
        JMP     @SALB       ;
SALB:   .BLK    1           ; "ALB" RETURN ADDRESS
;
;*****************************************************************************
;       A L B   A L B   A L B   A L B   A L B   A L B   A L B   A L B   A L B
;*****************************************************************************
;*****************************************************************************
;       A L N   A L N   A L N   A L N   A L N   A L N   A L N   A L N   A L N
;*****************************************************************************
;
; "ALN" PERFORMS ALPHANUMERIC("A" - "Z", "0" - "9", AND ".")
;       CHARACTER VERIFICATION
;
;       ENTRY:      ALN + 1
;       INPUT:      CHARACTER TO BE VERIFIED, IN ALN
;       OUTPUT:     VERIFICATION FLAG(0=INVALID,1=VALID), IN ALN
;       REQUIRED:   "ALB"
;
ALN:    .BLK    1           ; ON ENTRY, CHARACTER TO BE VERIFIED
                            ; ON EXIT, VERIFICATION FLAG(0=INVALID,1=VALID)
        STA     3,SALN      ; SAVE RETURN ADDRESS
        LDA     2,.ALB      ; CALL "ALB" TO CHECK IF THE CHARACTER IS ALPHABETIC
        LDA     0,ALN       ;
        STA     0,0,2       ;
        JSR     1,2         ;
        LDA     0,@.ALB     ;
        MOV#    0,0,SZR     ; CHECK THE "ALB" VERIFICATION FLAG
        JMP     ALN1        ; ALPHABETIC -- MARK THE CHARACTER AS VALID AND RETURN
```

```
        LDA     0,ALN       ; NONALPHABETIC -- CHECK IF THE CHARACTER IS "0" - "9"
        LDA     1,DIG0      ;
        LDA     2,DIG9      ;
        ADCZ#   2,0,SNC     ; SKIP IF AC0(CHARACTER) > AC2(ASCII "9")
        ADCZ#   0,1,SZC     ; SKIP IF AC0(CHARACTER) >= AC1(ASCII "0")
        JMP     ALN2        ; NOT "0" - "9" EITHER, SO MARK THE CHARACTER AS
                            ; INVALID AND RETURN
ALN1:   LDA     0,POS1      ; MARK THE CHARACTER AS VALID(ALN <-- 1)
                            ; AND RETURN
        STA     0,ALN       ;
        JMP     @SALN       ;
ALN2:   LDA     0,ZERO      ; MARK THE CHARACTER AS INVALID(ALN <-- 0)
                            ; AND RETURN
        STA     0,ALN       ;
        JMP     @SALN       ;
SALN:   .BLK    1           ; "ALN" RETURN ADDRESS
;
;*****************************************************************************
;       A L N   A L N   A L N   A L N   A L N   A L N   A L N   A L N   A L N
;*****************************************************************************
;*****************************************************************************
;       B C H K   B C H K   B C H K   B C H K   B C H K   B C H K   B C H K
;*****************************************************************************
;
; "BCHK" EVALUATES THE BREAKPOINTING ASSERTION
;
;       ENTRY:      BCHK + 1
;       INPUT:      NONE
;       OUTPUT:     VALUE OF BREAKPOINTING ASSERTION(0=FALSE,1=TRUE), IN BCHK
;       REQUIRED:   "PSE"
;
BCHK:   .BLK    1           ; ON EXIT, VALUE OF BREAKPOINTING
                            ; ASSERTION(0=FALSE,1=TRUE)
        STA     3,SBCHK     ; SAVE RETURN ADDRESS
        LDA     2,.PSE      ; EVALUATE THE BREAKPOINTING ASSERTION BY
```

```
                        ; CALLING "PSE" TO EVALUATE ITS POLISH STACK
        LDA     0,.BPS  ;
        STA     0,0,2   ;
        JSR     2,2     ;
        LDA     2,.PSE  ;
        LDA     0,1,2   ;
        STA     0,BCHK  ; BCHK <-- "PSE" EVALUATION FLAG
        JMP     @SBCHK  ; RETURN
SBCHK:  .BLK    1       ; "BCHK" RETURN ADDRESS
;
;*****************************************************************************
;       B C H K   B C H K   B C H K   B C H K   B C H K   B C H K   B C H K
;*****************************************************************************
;*****************************************************************************
;       B R K   B R K   B R K   B R K   B R K   B R K   B R K   B R K   B R K
;*****************************************************************************
;
; "BRK" DISPLAYS THE USER PROGRAM STATE AT A BREAKPOINT
;
;       ENTRY:      BRK
;       INPUT:      NONE
;       OUTPUT:     NONE
;       REQUIRED:   "DMP"
;
BRK:    STA     3,SBRK  ; SAVE RETURN ADDRESS
        LDA     2,.DMP  ; CALL "DMP" TO OUTPUT THE USER PROGRAM STATE
                        ; AT A BREAKPOINT(DMP <-- 1)
        LDA     0,POS1  ;
        STA     0,0,2   ;
        JSR     1,2     ;
        JMP     @SBRK   ; RETURN
SBRK:   .BLK    1       ; "BRK" RETURN ADDRESS
;
;*****************************************************************************
;       B R K   B R K   B R K   B R K   B R K   B R K   B R K   B R K   B R K
```

```
;*****************************************************************************
;*****************************************************************************
;       B 2 D   B 2 D   B 2 D   B 2 D   B 2 D   B 2 D   B 2 D   B 2 D   B 2 D
;*****************************************************************************
;
; "B2D" PERFORMS BINARY TO DECIMAL CONVERSION PER "INTRODUCTION TO
;       PROGRAMMING THE NOVA COMPUTERS", 093-000067-01, DATA GENERAL
;       CORPORATION, SOUTHBORO, MA, 1972, PAGE A-6
;
;        ENTRY:       B2D + 1
;        INPUT:       BINARY INTEGER TO BE CONVERTED, IN B2D
;        OUTPUT:      NONE
;        REQUIRED:    "PUTC"
;
B2D:    .BLK    1           ; ON ENTRY, BINARY INTEGER TO BE CONVERTED
        STA     3,SB2D      ; SAVE RETURN ADDRESS
        LDA     1,B2D       ; AC1 <-- BINARY INTEGER TO BE CONVERTED
        LDA     3,INST      ; SET UP LDA COMMAND
        STA     3,.+1       ;
B2D1:   .BLK    1           ; AC2 <-- POWER OF 10
        LDA     0,DIG0      ; AC0 <-- ASCII "0"
        SUBO    2,1,SNC     ; STILL PLUS IF NO CARRY
        INC     0,0,SKP     ; INCREMENT ASCII CHARACTER
        ADD     2,1,SKP     ; TOO MUCH -- ADD BACK
        JMP     .-3         ;
        JSR     @.PUTC      ; OUTPUT CHARACTER
        ISZ     B2D1        ; INCREMENT LDA COMMAND
        MOVR    2,2,SNC     ; CHECK FOR THE LAST DIGIT
        JMP     B2D1        ; NO -- CONTINUE
        JMP     @SB2D       ; YES -- RETURN
        .RDX    10          ; CHANGE RADIX TO 10
TENS:   10000
        1000
        100
        10
```

```
        1
        .RDX    8           ; CHANGE RADIX BACK TO 8
INST:   LDA     2,.+TENS-B2D1
SB2D:   .BLK    1           ; "B2D" RETURN ADDRESS
;
;*****************************************************************************
;         B 2 D   B 2 D   B 2 D   B 2 D   B 2 D   B 2 D   B 2 D   B 2 D   R 2 D
;*****************************************************************************
;*****************************************************************************
;         B 2 0   B 2 0   B 2 0   B 2 0   B 2 0   B 2 0   B 2 0   B 2 0   R 2 0
;*****************************************************************************
;
; "B20" PERFORMS BINARY TO OCTAL CONVERSION PER "INTRODUCTION TO
;       PROGRAMMING THE NOVA COMPUTERS", 093-000067-01, DATA GENERAL
;       CORPORATION, SOUTHBORO, MA, 1972, PAGE A-5
;
;        ENTRY:       B20 + 1
;        INPUT:       BINARY INTEGER TO BE CONVERTED, IN B20
;        OUTPUT:      NONE
;        REQUIRED:    "PUTC"

B20:    .BLK    1           ; ON ENTRY, BINARY INTEGER TO BE CONVERTED
        STA     3,SB20      ; SAVE THE RETURN ADDRESS
        LDA     1,B20       ; AC1 <-- BINARY INTEGER TO BE CONVERTED
        SUBZR   2,2         ; AC2 <-- 100000
B201:   LDA     0,DIG0      ; AC0 <-- ASCII "0"
        SUBO    2,1,SNC     ; STILL PLUS IF NO CARRY
        INC     0,0,SKP     ; INCREMENT ASCII CHARACTER
        ADD     2,1,SKP     ; TOO MUCH -- ADD BACK
        JMP     .-3         ;
        JSR     @.PUTC      ; OUTPUT ASCII CHARACTER
        MOVZR   2,2         ; SHIFT RIGHT THREE BITS
        MOVZR   2,2         ;
        MOVZR   2,2,SZR     ; CHECK FOR THE LAST DIGIT
        JMP     B201        ; NO -- CONTINUE
```

```
        JMP     @SB20       ; YES -- RETURN
SB20:   .BLK    1           ; "B20" RETURN ADDRESS
;
;*****************************************************************************
;       B 2 0   B 2 0   B 2 0   B 2 0   B 2 0   B 2 0   B 2 0   B 2 0   B 2 0
;*****************************************************************************
;*****************************************************************************
;       C B I T   C B I T   C B I T   C B I T   C B I T   C B I T   C B I T
;*****************************************************************************
;
; "CBIT" OUTPUTS THE VALUE OF AN ENCODED(ROTATED LEFT INTO BIT 15) CARRY BIT
;
;       ENTRY:      CBIT + 1
;       INPUT:      ENCODED CARRY BIT TO BE OUTPUT, IN CBIT
;       OUTPUT:     NONE
;       REQUIRED:   "PUTC"
;
CBIT:   .BLK    1           ; ON ENTRY, ENCODED CARRY BIT TO BE OUTPUT
        STA     3,SCBIT     ; SAVE RETURN ADDRESS
        LDA     0,SPACE     ; OUTPUT TWO SPACES
        JSR     @.PUTC      ;
        JSR     @.PUTC      ;
        LDA     0,DIG0      ; ASSUME THE CARRY BIT IS A 0(AC0 <-- ASCII "0")
        LDA     1,CBIT      ;
        MOVR#   1,1,SZC     ; CHECK IF THIS IS RIGHT
        LDA     0,DIG1      ; NO -- IT'S A 1(AC0 <-- ASCII "1")
        JSR     @.PUTC      ; OUTPUT AC0, THE ASCII EQUIVALENT OF THE ENCODED CARRY
        LDA     0,SPACE     ; OUTPUT TWO SPACES
        JSR     @.PUTC      ;
        JSR     @.PUTC      ;
        JMP     @SCBIT      ; RETURN
SCBIT:  .BLK    1           ; "CBIT" RETURN ADDRESS
;
;*****************************************************************************
;       C B I T   C B I T   C B I T   C B I T   C B I T   C B I T   C B I T
```

```
;*****************************************************************************
;*****************************************************************************
;        C P Y   C P Y   C P Y   C P Y   C P Y   C P Y   C P Y   C P Y   C P Y
;*****************************************************************************
;
; "CPY" COPIES THE CURRENT CPU STATE VECTOR TO THE PREVIOUS CPU STATE VECTOR
;
;         ENTRY:      CPY
;         INPUT:      NONE
;         OUTPUT:     NONE
;         REQUIRED:   NONE
;
CPY:    LDA     0,NEWA0     ; AC0
        STA     0,OLDA0     ;
        LDA     0,NEWA1     ; AC1
        STA     0,OLDA1     ;
        LDA     0,NEWA2     ; AC2
        STA     0,OLDA2     ;
        LDA     0,NEWA3     ; AC3
        STA     0,OLDA3     ;
        LDA     0,NEWCY     ; CARRY BIT
        STA     0,OLDCY     ;
        LDA     0,NEWPC     ; PROGRAM COUNTER
        STA     0,OLDPC     ;
        JMP     0,3         ; RETURN
;
;*****************************************************************************
;        C P Y   C P Y   C P Y   C P Y   C P Y   C P Y   C P Y   C P Y   C P Y
;*****************************************************************************
;*****************************************************************************
;        D M P   D M P   D M P   D M P   D M P   D M P   D M P   D M P   D M P
;*****************************************************************************
;
; "DMP" DUMPS THE CURRENT AND PREVIOUS CPU STATES OF THE USER PROGRAM
;
```

```
;         ENTRY:      DMP + 1
;         INPUT:      DUMP FLAG(1=BREAKPOINT,2=HALT,4=INTERRUPT), IN DMP
;         OUTPUT:     NONE
;         REQUIRED:   "B20", "CBIT", "PUTB", "SP3"
;
DMP:    .BLK    1           ; ON ENTRY, FLAG TO INDICATE TYPE
                            ; OF DUMP(1=BREAKPOINT,2=HALT,4=INTERRUPT)
        STA     3,SDMP      ; SAVE RETURN ADDRESS
        LDA     2,.PUTB     ; OUTPUT DUMP HEADINGS
        LDA     0,.OUT7     ;
        STA     0,0,2       ;
        JSR     1,2         ;
        LDA     2,.PUTB     ; OUTPUT CAUSE OF DUMP
        LDA     1,DMP       ;
        MOVR    1,1,SZC     ; CHECK IF IT WAS A BREAKPOINT(DMP=1)
        LDA     0,.OUT9     ; YES -- A BREAKPOINT
        MOVR    1,1,SZC     ; CHECK IF IT WAS A HALT(DMP=2)
        LDA     0,.OUT8     ; YES -- A HALT
        MOVR    1,1,SZC     ; CHECK IF IT WAS AN INTERRUPT(DMP=4)
        LDA     0,.OUT6     ; YES -- AN INTERRUPT
        STA     0,0,2       ;
        JSR     1,2         ; OUTPUT APPROPRIATE LABEL FOR CAUSE OF DUMP
        LDA     2,.B20      ; OUTPUT CURRENT ADDRESS
        LDA     0,NEWPC     ;
        STA     0,0,2       ;
        JSR     1,2         ;
        JSR     @.SP3       ;
        LDA     2,.B20      ; OUTPUT CURRENT INSTRUCTION
        LDA     0,@NEWPC    ;
        STA     0,0,2       ;
        JSR     1,2         ;
        JSR     @.SP3       ;
        LDA     2,.CBIT     ; OUTPUT CURRENT CARRY BIT
        LDA     0,NEWCY     ;
        STA     0,0,2       ;
```

```
JSR     1,2         ;
JSR     @.SP3       ;
LDA     2,.B20      ; OUTPUT CURRENT AC0
LDA     0,NEWA0     ;
STA     0,0,2       ;
JSR     1,2         ;
JSR     @.SP3       ;
LDA     2,.B20      ; OUTPUT CURRENT AC1
LDA     0,NEWA1     ;
STA     0,0,2       ;
JSR     1,2         ;
JSR     @.SP3       ;
LDA     2,.B20      ; OUTPUT CURRENT AC2
LDA     0,NEWA2     ;
STA     0,0,2       ;
JSR     1,2         ;
JSR     @.SP3       ;
LDA     2,.B20      ; OUTPUT CURRENT AC3
LDA     0,NEWA3     ;
STA     0,0,2       ;
JSR     1,2         ;
LDA     0,PDUMP     ;
MOV#    0,0,SNR     ; CHECK PDUMP TO SEE IF ANY INSTRUCTIONS ACTUALLY
                    ; GOT EXECUTED
JMP     @SDMP       ; NO -- THERE WAS NO PREVIOUS CPU STATE, SO JUST RETURN
LDA     2,.PUTB     ; YES -- OUTPUT LABEL FOR PREVIOUS CPU STATE LINE
LDA     0,.OUTA     ;
STA     0,0,2       ;
JSR     1,2         ;
LDA     2,.B20      ; OUTPUT PREVIOUS ADDRESS
LDA     0,OLDPC     ;
STA     0,0,2       ;
JSR     1,2         ;
JSR     @.SP3       ;
LDA     2,.B20      ; OUTPUT PREVIOUS INSTRUCTION
```

```
        LDA     0,@OLDPC    ;
        STA     0,0,2       ;
        JSR     1,2         ;
        JSR     @.SP3       ;
        LDA     2,.CBIT     ; OUTPUT PREVIOUS CARRY BIT
        LDA     0,OLDCY     ;
        STA     0,0,2       ;
        JSR     1,2         ;
        JSR     @.SP3       ;
        LDA     2,.B20      ; OUTPUT PREVIOUS AC0
        LDA     0,OLDA0     ;
        STA     0,0,2       ;
        JSR     1,2         ;
        JSR     @.SP3       ;
        LDA     2,.B20      ; OUTPUT PREVIOUS AC1
        LDA     0,OLDA1     ;
        STA     0,0,2       ;
        JSR     1,2         ;
        JSR     @.SP3       ;
        LDA     2,.B20      ; OUTPUT PREVIOUS AC2
        LDA     0,OLDA2     ;
        STA     0,0,2       ;
        JSR     1,2         ;
        JSR     @.SP3       ;
        LDA     2,.B20      ; OUTPUT PREVIOUS AC3
        LDA     0,OLDA3     ;
        STA     0,0,2       ;
        JSR     1,2         ;
        JMP     @SDMP       ; RETURN
SDMP:   .BLK    1           ; "DMP" RETURN ADDRESS
;
;*****************************************************************************
;       D M P   D M P   D M P   D M P   D M P   D M P   D M P   D M P   D M P
;*****************************************************************************
;*****************************************************************************
```

```
;       E C A C   E C A C   E C A C   E C A C   E C A C   E C A C   E C A C
;*****************************************************************************
;
; "ECAC" EXAMINES THE CHARACTER IN ATXT, THE ASSERTION TEXT BUFFER,
;       CURRENTLY POINTED TO BY ATXTP, THE ASSERTION TEXT POINTER
;
;       ENTRY:      ECAC
;       INPUT:      NONE
;       OUTPUT:     CURRENT ASSERTION TEXT CHARACTER, IN AC0
;       REQUIRED:   NONE
;       NOTE:       ATXTP, THE ASSERTION TEXT POINTER, IS NOT ADVANCED
;
ECAC:  LDA     2,.ATXT     ; AC2 <-- STARTING ADDRESS OF ASSERTION TEXT BUFFFR
       LDA     1,ATXTP     ; AC1 <-- ASSERTION TEXT POINTER
       ADD     1,2         ; ADD THESE TWO TO OFFSET TO THE CURRENT CHARACTER
       LDA     0,0,2       ; AC0 <-- CURRENT ASSERTION TEXT CHARACTER
       JSR     0,3         ; RETURN
;
;*****************************************************************************
;       E C A C   E C A C   E C A C   E C A C   E C A C   E C A C   E C A C
;*****************************************************************************
;*****************************************************************************
;       E N A C   E N A C   E N A C   E N A C   E N A C   E N A C   E N A C
;*****************************************************************************
;
; "ENAC" EXAMINES THE CHARACTER IN ATXT, THE ASSERTION TEXT BUFFER,
;       FOLLOWING THE ONE CURRENTLY POINTED TO BY ATXTP, THE ASSERTION
;       TEXT POINTER
;
;       ENTRY:      ENAC
;       INPUT:      NONE
;       OUTPUT:     NEXT ASSERTION TEXT CHARACTER, IN AC0
;       REQUIRED:   NONE
;       NOTE:       ATXTP, THE ASSERTION TEXT POINTER, IS NOT ADVANCED
;
```

```
ENAC:   LDA     2,.ATXT     ; AC2 <-- STARTING ADDRESS OF ASSERTION TEXT BUFFER
        LDA     1,ATXTP     ; AC1 <-- ASSERTION TEXT POINTER
        INC     1,1         ; INCREMENT THE ASSERTION TEXT POINTER
        ADD     1,2         ; ADD THESE TWO TO OFFSET TO THE NEXT CHARACTER
        LDA     0,0,2       ; AC0 <-- NEXT ASSERTION TEXT CHARACTER
        JMP     0,3         ; RETURN
;
;*****************************************************************************
;       E N A C   E N A C   E N A C   E N A C   E N A C   E N A C   E N A C
;*****************************************************************************
;*****************************************************************************
;       G A T   G A T   G A T   G A T   G A T   G A T   G A T   G A T   G A T
;*****************************************************************************
;
; "GAT" PERFORMS TELETYPE INPUT OF THE ASSERTION TEXT
;
;       ENTRY:      GAT
;       INPUT:      NONE
;       OUTPUT:     NONE
;       REQUIRED:   "GETC", "PUTB", "PUTC", "SP3"
;
GAT:    STA     3,SGAT      ; SAVE RETURN ADDRESS
        LDA     0,NEG1      ; MOVE THE ASSERTION TEXT POINTER BACK TO
                            ; THE FRONT OF THE BUFFER(ATXTP <-- -1)
        STA     0,ATXTP     ;
GAT1:   JSR     @.GETC      ; AC0 <-- NEXT INPUT CHARACTER
        JSR     @.PUTC      ; ECHO IT
        LDA     1,CR        ;
        SUB#    0,1,SZR     ; CHECK IF IT IS A CARRIAGE RETURN
        JMP     GAT2        ; NOT A CARRIAGE RETURN
        LDA     0,LF        ; YES -- OUTPUT A LINE FEED AND TAB IN FOR
                            ; ANOTHER LINE OF INPUT
        JSR     @.PUTC      ;
        JSR     @.SP3       ;
        LDA     0,SPACE     ;
```

```
        JSR     @.PUTC      ;
        JSR     @.PUTC      ;
        JMP     GAT1        ;
GAT2:   LDA     1,SPACE     ;
        SUB#    0,1,SNR     ; NOT A CARRIAGE RETURN -- CHECK IF IT IS A BLANK
        JMP     GAT1        ; YES -- IGNORE IT
        LDA     1,ARROW     ;
        SUB#    0,1,SZR     ; NOT A BLANK EITHER -- CHECK IF IT IS
                            ; A BACKWARDS ARROW
        JMP     GAT3        ; NOT A BACKWARDS ARROW
        LDA     0,NEG1      ; YES -- CHECK IF THE POINTER CAN BE BACKED UP
        LDA     1,ATXTP     ;
        SUB#    0,1,SNR     ;
        JMP     GAT1        ; IT'S ALREADY AT THE FRONT, SO LEAVE IT ALONE
        ADD     0,1         ; NOT AT THE FRONT(ATXTP > -1), SO BACK
                            ; IT UP(ATXTP <-- ATXTP - 1)
        STA     1,ATXTP     ;
        JMP     GAT1        ;
GAT3:   LDA     1,ATXTP     ; THE CHARACTER WAS NOT A CARRIAGE RETURN, A BLANK OR
                            ; A BACKWARDS ARROW, SO ADD IT TO THE END OF THE
                            ; BUFFER -- IF THERE'S ROOM LEFT
        INC     1,1         ; AC1 <-- ATXTP + 1
        LDA     2,ATXTM     ; AC2 <-- ATXTM(MAXIMUM VALUE OF ATXTP)
        ADCZ#   2,1,SZC     ; SKIP IF AC1(ATXTP + 1) <= AC2(ATXTM)
        JMP     GAT4        ; ATXT OVERFLOW
        STA     1,ATXTP     ; THERE WAS ROOM, SO ADD THE CHARACTER AND
                            ; UPDATE ATXTP(ATXTP <-- ATXTP + 1)
        LDA     2,.ATXT     ;
        ADD     1,2         ;
        LDA     1,SCOLN     ; TRANSLATE ";" TO "&"
        SUB#    0,1,SNR     ;
        LDA     0,AMPER     ;
        LDA     1,COMMA     ; TRANSLATE "," TO ":"
        SUB#    0,1,SNR     ;
        LDA     0,COLON     ;
```

```
        STA     0,0,2       ;
        LDA     1,DSIGN     ;
        SUB#    0,1,SZR     ; CHECK IF THAT LAST CHARACTER WAS A FENCE("$")
        JMP     GAT1        ; NO -- GO GET ANOTHER CHARACTER
        JMP     @SGAT       ; YES -- RETURN
GAT4:   LDA     2,.PUTB     ; OUTPUT A FATAL ERROR MESSAGE(ATXT OVERFLOW) AND
                            ; RETURN TO "ALA" DIRECTLY
        LDA     0,.ERR6     ;
        STA     0,0,2       ;
        JSR     1,2         ;
        JMP     @.ALA1      ;
SGAT:   .BLK    1           ; "GAT" RETURN ADDRESS
;
;*****************************************************************************
;       G A T   G A T   G A T   G A T   G A T   G A T   G A T   G A T   G A T
;*****************************************************************************
;*****************************************************************************
;       G E T C   G E T C   G E T C   G E T C   G E T C   G E T C   G E T C
;*****************************************************************************
;
; "GETC" PERFORMS SINGLE CHARACTER TELETYPE INPUT
;
;       ENTRY:      GETC
;       INPUT:      NONE
;       OUTPUT:     TELETYPE INPUT CHARACTER, IN AC0
;       REQUIRED:   NONE
;
GETC:   NIOS    TTI         ; START TELETYPE
        SKPDN   TTI         ; WAIT FOR INPUT
        JMP     .-1         ;
        DIAC    0,TTI       ; AC0 <-- CHARACTER AND CLEAR TELETYPE
        LDA     1,CMASK     ; AC1 <-- 000177(ASCII CHARACTER MASK)
        AND     1,0         ; MASK OFF THE PARITY BIT
        JMP     0,3         ; RETURN
;
```

```
;*****************************************************************************
;       G E T C   G E T C   G E T C   G E T C   G E T C   G E I C   G E T C
;*****************************************************************************
;*****************************************************************************
;       H L T   H L T   H L T   H L T   H L T   H L T   H L T   H L T   H L T
;*****************************************************************************
;
; "HLT" DISPLAYS THE USER PROGRAM STATE AT A HALT
;
;       ENTRY:      HLT
;       INPUT:      NONE
;       OUTPUT:     NONE
;       REQUIRED:   "DMP"
;
HLT:    STA     3,SHLT      ; SAVE RETURN ADDRESS
        LDA     2,.DMP      ; CALL "DMP" TO OUTPUT THE USER PROGRAM STATE
                            ; AT A HALT(DMP <-- 2)
        LDA     0,POS2      ;
        STA     0,0,2       ;
        JSR     1,2         ;
        JSR     @.CPY       ; NOW, "EXECUTE" THE HALT -- CALL "CPY" TO COPY
                            ; THE CURRENT CPU STATE VECTOR TO THE PREVIOUS CPU
                            ; STATE VECTOR
        LDA     0,OLDPC     ; FINALLY, NEWPC <-- OLDPC + 1
        INC     0,0         ;
        LDA     1,AMASK     ;
        AND     0,1         ;
        STA     1,NEWPC     ;
        JMP     @SHLT       ; RETURN
SHLT:   .BLK    1           ; "HLT" RETURN ADDRESS
;
;*****************************************************************************
;       H L T   H L T   H L T   H L T   H L T   H L T   H L T   H L T   H L T
;*****************************************************************************
;*****************************************************************************
```

```
;        N T K N   N T K N   N T K N   N T K N   N T K N   N T K N   N T K N
;*****************************************************************************
;
; "NTKN" RETURNS THE NEXT TOKEN OF THE ASSERTION INPUT STRING
;
;        ENTRY:      NTKN + 3
;        INPUT:      NONE
;        OUTPUT:     TOKEN CLASS OF NEXT TOKEN, IN NTKN
;                    IF TOKEN IS OPERAND --
;                                OPERAND TYPE, IN TVAL1(NTKN + 1)
;                                OPERAND VALUE, IN TVAL2(NTKN + 2)
;                    IF TOKEN IS OPERATOR --
;                                OPERATOR ID, IN TVAL1(NTKN + 1)
;                                OPERATOR PRIORITY, IN TVAL2(NTKN + 2)
;        REQUIRED:   "ALB", "ALN", "ECAC", "ENAC", "OCT", "RNAC", "SRCH"
;
NTKN:   .BLK    1           ; ON EXIT, TOKEN CLASS OF NEXT TOKEN
TVAL1:  .BLK    1           ; ON EXIT, OPERAND TYPE/OPERATOR ID
TVAL2:  .BLK    1           ; ON EXIT, OPERAND VALUE/OPERATOR PRIORITY
        STA     3,SNTKN     ; SAVE RETURN ADDRESS
        JSR     @.RNAC      ; AC0 <-- NEXT ASSERTION TEXT CHARACTER
        LDA     1,DSIGN     ;
        SUB#    0,1,SZR     ; CHECK IF IT IS A "$"(FENCE)
        JMP     NTKN1       ; NOT A "$"
        LDA     0,POS5      ;
        STA     0,NTKN      ; YES -- TOKEN CLASS <-- 5
        JMP     @SNTKN      ; RETURN
NTKN1:  LDA     1,LPARN     ;
        SUB#    0,1,SZR     ; CHECK IF IT IS A "("
        JMP     NTKN2       ; NOT A "("
        LDA     0,POS1      ;
        STA     0,NTKN      ; YES -- TOKEN CLASS <-- 1
        LDA     0,POS10     ;
        STA     0,TVAL1     ; OPERATOR ID <-- 10
        LDA     0,ZERO      ;
```

```
        STA     0,TVAL2     ; OPERATOR PRIORITY <-- 0
        JMP     @SNTKN      ; RETURN
NTKN2:  LDA     1,RPARN     ;
        SUB#    0,1,SZR     ; CHECK IF IT IS A ")"
        JMP     NTKN3       ; NOT A ")"
        LDA     0,POS2      ;
        STA     0,NTKN      ; YES -- TOKEN CLASS <-- 2
        LDA     0,POS11     ;
        STA     0,TVAL1     ; OPERATOR ID <-- 11
        LDA     0,ZERO      ;
        STA     0,TVAL2     ; OPERATOR PRIORITY <-- 0
        JMP     @SNTKN      ; RETURN
NTKN3:  LDA     1,EQ        ;
        SUB#    0,1,SZR     ; CHECK IF IT IS AN "="
        JMP     NTKN4       ; NOT AN "="
        LDA     0,POS3      ;
        STA     0,NTKN      ; YES -- TOKEN CLASS <-- 3
        LDA     0,POS2      ;
        STA     0,TVAL1     ; OPERATOR ID <-- 2
        LDA     0,POS3      ;
        STA     0,TVAL2     ; OPERATOR PRIORITY <-- 3
        JMP     @SNTKN      ; RETURN
NTKN4:  LDA     1,PSIGN     ;
        SUB#    0,1,SZR     ; CHECK IF IT IS A "+"
        JMP     NTKN5       ; NOT A "+"
        LDA     0,POS3      ;
        STA     0,NTKN      ; YES -- TOKEN CLASS <-- 3
        LDA     0,ZERO      ;
        STA     0,TVAL1     ; OPERATOR ID <-- 0
        LDA     0,POS4      ;
        STA     0,TVAL2     ; OPERATOR PRIORITY <-- 4
        JMP     @SNTKN      ; RETURN
NTKN5:  LDA     1,MSIGN     ;
        SUB#    0,1,SZR     ; CHECK IF IT IS A "-"
        JMP     NTKN6       ; NOT A "-"
```

```
        LDA     0,POS3      ;
        STA     0,NTKN      ; YES -- TOKEN CLASS <-- 5
        LDA     0,POS1      ;
        STA     0,TVAL1     ; OPERATOR ID <-- 1
        LDA     0,POS4      ;
        STA     0,TVAL2     ; OPERATOR PRIORITY <-- 4
        JMP     @SNTKN      ; RETURN
NTKN6:  LDA     1,COLON     ;
        SUB#    0,1,SZR     ; CHECK IF IT IS A ":"
        JMP     NTKN7       ; NOT A ":"
        LDA     0,POS3      ;
        STA     0,NTKN      ; YES -- TOKEN CLASS <-- 3
        LDA     0,POS9      ;
        STA     0,TVAL1     ; OPERATOR ID <-- 9
        LDA     0,POS1      ;
        STA     0,TVAL2     ; OPERATOR PRIORITY <-- 1
        JMP     @SNTKN      ; RETURN
NTKN7:  LDA     1,AMPER     ;
        SUB#    0,1,SZR     ; CHECK IF IT IS AN "&"
        JMP     NTKN8       ; NOT AN "&"
        LDA     0,POS3      ;
        STA     0,NTKN      ; YES -- TOKEN CLASS <-- 5
        LDA     0,POS8      ;
        STA     0,TVAL1     ; OPERATOR ID <-- 8
        LDA     0,POS2      ;
        STA     0,TVAL2     ; OPERATOR PRIORITY <-- 2
        JMP     @SNTKN      ; RETURN
NTKN8:  LDA     1,LT        ;
        SUB#    0,1,SZR     ; CHECK IF IT IS A "<"
        JMP     NTKNA       ; NOT A "<"
        LDA     0,POS3      ;
        STA     0,NTKN      ; YES -- TOKEN CLASS <-- 5
        LDA     0,POS4      ;
        STA     0,TVAL1     ; ASSUME JUST A "<" -- OPERATOR ID <-- 4
        LDA     0,POS3      ;
```

```
        STA     0,TVAL2     ; OPERATOR PRIORITY <-- 3
        JSR     @.ENAC      ; LOOK AT THE CHARACTER FOLLOWING IT
        LDA     1,GT        ;
        SUB#    0,1,SZR     ; CHECK IF IT IS A ">"
        JMP     NTKN9       ;
        LDA     0,POS3      ;
        STA     0,TVAL1     ; YES -- TOKEN IS "<>" -- OPERATOR ID <-- 3
        JSR     @.RNAC      ; MOVE UP THE POINTER BECAUSE OF THE ">"
        JMP     @SNTKN      ; RETURN
NTKN9:  LDA     1,EQ        ;
        SUB#    0,1,SZR     ; NO -- CHECK IF IT IS AN "="
        JMP     @SNTKN      ; NO -- JUST A "<" AS ASSUMED -- RETURN
        LDA     0,POS6      ;
        STA     0,TVAL1     ; YES -- TOKEN IS "<=" -- OPERATOR ID <-- 6
        JSR     @.RNAC      ; MOVE UP THE POINTER BECAUSE OF THE "="
        JMP     @SNTKN      ; RETURN
NTKNA:  LDA     1,GT        ;
        SUB#    0,1,SZR     ; CHECK IF IT IS A ">"
        JMP     NTKNB       ; NOT A ">"
        LDA     0,POS3      ;
        STA     0,NTKN      ; YES -- TOKEN CLASS <-- 3
        LDA     0,POS5      ;
        STA     0,TVAL1     ; ASSUME JUST A ">" -- OPERATOR ID <-- 5
        LDA     0,POS3      ;
        STA     0,TVAL2     ; OPERATOR ID <-- 3
        JSR     @.ENAC      ; LOOK AT THE CHARACTER FOLLOWING IT
        LDA     1,EQ        ;
        SUB#    0,1,SZR     ; CHECK IF IT IS AN "="
        JMP     @SNTKN      ; NO -- JUST A ">" AS ASSUMED -- RETURN
        LDA     0,POS7      ;
        STA     0,TVAL1     ; YES -- TOKEN IS ">=" -- OPERATOR ID <-- 7
        JSR     @.RNAC      ; MOVE UP THE POINTER BECAUSE OF THE "="
        JMP     @SNTKN      ; RETURN
NXCD1:  .BLK    1           ; COUNTER TO LIMIT THE OCTAL STRING TO SIX DIGITS
SNTKN:  .BLK    1           ; "NTKN" RETURN ADDRESS
```

```
.TVL1:  TVAL1
.TVL2:  TVAL2
NTKNB:  LDA     2,.OCT      ; CALL "OCT" TO SEE IF IT'S AN OCTAL DIGIT(LITERAL)
        STA     0,0,2       ;
        JSR     1,2         ;
        LDA     0,@.OCT     ;
        MOVL#   0,0,SZC     ; CHECK THE "OCT" VERIFICATION FLAG
        JMP     NTKNF       ; NOT AN OCTAL DIGIT, SO NOT A LITERAL
        STA     0,LVAL      ; LVAL <-- VALUE OF FIRST DIGIT
        LDA     0,POS6      ;
        STA     0,MXOD1     ; MXOD1 <-- 6
NTKNC:  JSR     @.ENAC      ; LOOK AT THE NEXT CHARACTER
        LDA     2,.OCT      ; CALL "OCT" TO SEE IF IT'S AN OCTAL DIGIT
        STA     0,0,2       ;
        JSR     1,2         ;
        LDA     0,@.OCT     ;
        MOVL#   0,0,SZC     ; CHECK THE "OCT" VERIFICATION FLAG
        JMP     NTKNE       ; NO -- THAT'S THE END OF THE LITERAL
        DSZ     MXOD1       ; YES -- CHECK IF THAT'S MORE THAN SIX DIGITS
        JMP     NTKND       ; NO -- ADD IT INTO THE PREVIOUS ACCUMULATED VALUE
        LDA     0,ZERO      ; YES -- TOO MANY DIGITS -- TOKEN CLASS <-- 0
        STA     0,@.NTKN    ;
        JMP     @SNTKN      ; RETURN
NTKND:  LDA     1,LVAL      ; MULTIPLY THE OLD ACCUMULATED LITERAL VALUE BY 8
                            ; AND ADD IN THE LATEST DIGIT INPUT
        ADDZL   1,1         ;
        MOVZL   1,1         ;
        ADD     0,1         ;
        STA     1,LVAL      ;
        JSR     @.RNAC      ; MOVE THE POINTER UP FOR THE LAST DIGIT PROCESSED
        JMP     NTKNC       ; GO GET ANOTHER CHARACTER
NTKNE:  LDA     0,POS4      ;
        STA     0,@.NTKN    ; LITERAL -- TOKEN CLASS <-- 4
        LDA     0,POS4      ;
        STA     0,@.TVL1    ; OPERAND TYPE <-- 4
```

```
        LDA     0,LVAL      ;
        STA     0,@.TVL2    ; OPERAND VALUE <-- LVAL
        JMP     @SNTKN      ; RETURN
NTKNF:  JSR     @.ECAC      ; THIS CHARACTER MUST BEGIN AN IDENTIFIER --
                            ; CALL "ALB" TO CHECK IF IT'S ALPHABETIC
        LDA     2,.ALB      ;
        STA     0,0,2       ;
        JSR     1,2         ;
        LDA     0,@.ALB     ;
        MOV#    0,0,SZR     ; CHECK THE "ALB" VERIFICATION FLAG
        JMP     NTKNG       ; YES -- IT'S ALPHABETIC -- PROCESS AN IDENTIFIER
        JSR     @.ECAC      ; NO -- MUST BE AN "*"(RESERVED SYMBOL)
        LDA     1,ASTRX     ;
        SUB#    0,1,SNR     ; CHECK IF IT IS AN "*"
        JMP     NTKNG       ; YES -- PROCESS AN IDENTIFIER
        LDA     0,ZERO      ;
        STA     0,@.NTKN    ; NO -- TOKEN CLASS <-- 0
        JMP     @SNTKN      ; RETURN
NTKNG:  JSR     @.ECAC      ;
        MOVS    0,0         ;
        STA     0,IVAL1     ; MOVE THIS FIRST IDENTIFIER CHARACTER INTO THE
                            ; LEFT BYTE OF IVAL1, AND FILL OUT THE REST OF
                            ; THE IDENTIFIER WITH NULLS
        LDA     0,ZERO      ;
        STA     0,IVAL2     ;
        STA     0,IVAL3     ;
        JSR     @.ENAC      ; LOOK AT THE NEXT CHARACTER
        LDA     2,.ALN      ; CALL "ALN" TO CHECK IF IT'S ALPHANUMERIC
        STA     0,0,2       ;
        JSR     1,2         ;
        LDA     0,@.ALN     ;
        MOV#    0,0,SNR     ; CHECK THE "ALN" VERIFICATION FLAG
        JMP     NTKNH       ; NOT ALPHANUMERIC -- END OF THE IDENTIFIER
        JSR     @.RNAC      ; ALPHANUMERIC -- MOVE THE POINTER UP
        LDA     1,IVAL1     ;
```

```
        ADD     0,1         ;
        STA     1,IVAL1     ; MOVE THIS CHARACTER INTO THE RIGHT BYTE OF IVAL1
        JSR     @.ENAC      ; LOOK AT THE NEXT CHARACTER
        LDA     2,.ALN      ; CALL "ALN" TO CHECK IF IT'S ALPHANUMERIC
        STA     0,0,2       ;
        JSR     1,2         ;
        LDA     0,@.ALN     ;
        MOV#    0,0,SNR     ; CHECK THE "ALN" VERIFICATION FLAG
        JMP     NTKNH       ; NOT ALPHANUMERIC -- END OF THE IDENTIFIER
        JSR     @.RNAC      ; ALPHANUMERIC -- MOVE THE POINTER UP
        MOVS    0,0         ;
        STA     0,IVAL2     ; MOVE THIS CHARACTER INTO THE LEFT BYTE OF IVAL2
        JSR     @.ENAC      ; LOOK AT THE NEXT CHARACTER
        LDA     2,.ALN      ; CALL "ALN" TO CHECK IF IT'S ALPHANUMERIC
        STA     0,0,2       ;
        JSR     1,2         ;
        LDA     0,@.ALN     ;
        MOV#    0,0,SNR     ; CHECK THE "ALN" VERIFICATION FLAG
        JMP     NTKNH       ; NOT ALPHANUMERIC -- END OF THE IDENTIFIER
        JSR     @.RNAC      ; ALPHANUMERIC -- MOVE THE POINTER UP
        LDA     1,IVAL2     ;
        ADD     0,1         ;
        STA     1,IVAL2     ; MOVE THIS CHARACTER INTO THE RIGHT BYTE OF IVAL2
        JSR     @.ENAC      ; LOOK AT THE NEXT CHARACTER
        LDA     2,.ALN      ; CALL "ALN" TO CHECK IF IT'S ALPHANUMERIC
        STA     0,0,2       ;
        JSR     1,2         ;
        LDA     0,@.ALN     ;
        MOV#    0,0,SNR     ; CHECK THE "ALN" VERIFICATION FLAG
        JMP     NTKNH       ; NOT ALPHANUMERIC -- END OF THE IDENTIFIER
        JSR     @.RNAC      ; ALPHANUMERIC -- MOVE THE POINTER UP
        MOVS    0,0         ;
        STA     0,IVAL3     ; MOVE THIS CHARACTER INTO THE LEFT BYTE OF IVAL3
NTKNH:  LDA     2,.SRCH     ; NOW, SEARCH FOR THE IDENTIFIER IN THE SYMBOL TABLE
        LDA     0,.IVAL     ;
```

```
        STA     0,0,2       ;
        JSR     1,2         ;
        LDA     0,@.SRCH    ;
        MOVL#   0,0,SNC     ; CHECK THE ABSOLUTE ADDRESS RETURNED
        JMP     NTKNI       ;
        LDA     0,ZERO      ;
        STA     0,@.NTKN    ; INVALID IDENTIFIER -- TOKEN CLASS <-- 0
        JMP     @SNTKN      ; RETURN
NTKNI:  STA     0,@.TVL2    ; VALID IDENTIFIER -- OPERAND VALUE <-- ABSOLUTE ADDRESS
        LDA     0,POS4      ;
        STA     0,@.NTKN    ; TOKEN CLASS <-- 4
        LDA     0,POS2      ;
        STA     0,@.TVL1    ; OPERAND TYPE <-- 2
        JMP     @SNTKN      ; RETURN
IVAL1:  .BLK    1           ; IDENTIFIER LABEL -- CHARACTERS 1 AND 2
IVAL2:  .BLK    1           ; IDENTIFIER LABEL -- CHARACTERS 3 AND 4
IVAL3:  .BLK    1           ; IDENTIFIER LABEL -- CHARACTER 5 AND "<00>"
IVAL:   .BLK    1           ; BINARY VALUE OF LITERAL
.TVAL:  IVAL1               ; ADDRESS OF IDENTIFIER LABEL
;
;*****************************************************************************
;       N T K N   N T K N   N T K N   N T K N   N T K N   N T K N   N T K N
;*****************************************************************************
;*****************************************************************************
;       O C T   O C T   O C T   O C T   O C T   O C T   O C T   O C T   O C T
;*****************************************************************************
;
; "OCT" PERFORMS OCTAL DIGIT VERIFICATION
;
;       ENTRY:      OCT + 1
;       INPUT:      CHARACTER TO BE VERIFIED, IN OCT
;       OUTPUT:     BINARY VALUE(0 - 7 -- -1 IF INVALID) OF CHARACTER, IN OCT
;       REQUIRED:   NONE
;
OCT:    .BLK    1           ; ON ENTRY, CHARACTER TO BE VERIFIED
```

```
                        ; ON EXIT, BINARY VALUE(0 - 7) OF OCTAL DIGIT, IF VALID
                        ;          -1, IF INVALID
        STA     3,SOCT  ; SAVE RETURN ADDRESS
        LDA     0,OCT   ; AC0 <-- CHARACTER TO BE VERIFIED
        LDA     1,DIG0  ;
        LDA     2,DIG7  ;
        ADCZ#   2,0,SNC ; SKIP IF AC0(CHARACTER) > AC2(ASCII "7")
        ADCZ#   0,1,SZC ; SKIP IF AC0(CHARACTER) >= AC1(ASCII "0")
        JMP     OCT1    ; NOT A VALID OCTAL DIGIT
        LDA     1,OMASK ; AN OCTAL DIGIT -- FIND ITS BINARY VALUE
                        ; BY MASKING OFF ALL BUT BITS 13-15 AND RETURN
        AND     1,0     ;
        STA     0,OCT   ;
        JMP     @SOCT   ;
OCT1:   LDA     0,NEG1  ; MARK THE CHARACTER AS AN INVALID
                        ; OCTAL DIGIT AND RETURN
        STA     0,OCT   ;
        JMP     @SOCT   ;
SOCT:   .BLK    1       ; "OCT" RETURN ADDRESS
;
;*****************************************************************************
;       O C T   O C T   O C T   O C T   O C T   O C T   O C T   O C T   O C T
;*****************************************************************************
;*****************************************************************************
;       0 2 B   0 2 B   0 2 B   0 2 B   0 2 B   0 2 B   0 2 B   0 2 B   0 2 B
;*****************************************************************************
;
; "02B" PERFORMS OCTAL TO BINARY CONVERSION
;
;       ENTRY:      02B + 3
;       INPUT:      NONE
;       OUTPUT:     OCTAL STRING VALIDITY FLAG(0=INVALID,1=VALID), IN 02B
;                   BINARY VALUE OF THE OCTAL STRING(IF VALID),
;                        IN BVAL(02B + 1)
;                   THE DELIMITING CHARACTER FOLLOWING THE END OF THE
```

```
;                                   STRING, IN NCHAR(O2B + 2)
;         REQUIRED:    "GETC", "OCT", "PUTC"
;
O2B:    .BLK    1           ; ON EXIT, STRING VALIDITY FLAG(0=INVALID,1=VALID)
BVAL:   .BLK    1           ; ON EXIT, ACCUMULATED BINARY VALUE OF THE STRING
NCHAR:  .BLK    1           ; ON EXIT, THE CHARACTER FOLLOWING THE END OF THE STRING
        STA     3,SO2B      ; SAVE RETURN ADDRESS
        LDA     0,ZERO      ; ZERO THE ACCUMULATED BINARY VALUE OF THE STRING
        STA     0,BVAL      ;
        STA     0,O2B       ; ASSUME THE STRING IS INVALID(O2B <-- 0)
        LDA     0,POS7      ; MXOD2 <-- MAXIMUM NUMBER OF OCTAL DIGITS
        STA     0,MXOD2     ;
O2E1:   JSR     @.GETC      ; NCHAR <-- NEXT CHARACTER INPUT
        STA     0,NCHAR     ;
        LDA     2,.OCT      ; CALL "OCT" TO CHECK IF THE NEXT CHARACTER
                            ; IS AN OCTAL DIGIT
        STA     0,0,2       ;
        JSR     1,2         ;
        LDA     0,@.OCT     ;
        MOVL#   0,0,SZC     ; CHECK THE "OCT" VERIFICATION FLAG
        JMP     @SO2B       ; NOT AN OCTAL DIGIT -- THAT'S ALL OF THE STRING
        DSZ     MXOD2       ; AN OCTAL DIGIT -- CHECK IF THAT'S MORE THAN SIX DIGITS
        JMP     O2B2        ;
        LDA     0,ZERO      ; YES -- MARK THE STRING AS INVALID AND RETURN
        STA     0,O2B       ;
        JMP     @SO2B       ;
O2B2:   LDA     1,BVAL      ; NO -- MULTIPLY THE OLD ACCUMULATED BINARY VALUE
                            ; BY 8(SHIFT LEFT 3 BITS) AND ADD IN THE LATEST
                            ; OCTAL DIGIT INPUT
        ADDZL   1,1         ;
        MOVZL   1,1         ;
        ADD     0,1         ;
        STA     1,BVAL      ;
        LDA     0,POS1      ; PROCESSED AT LEAST ONE OCTAL DIGIT, SO MARK
                            ; THE STRING AS VALID FOR NOW(O2B <-- 1)
```

```
        STA     0,02B       ;
        LDA     0,NCHAR     ; ECHO THE LATEST OCTAL DIGIT INPUT
        JSR     @.PUTC      ;
        JMP     02B1        ; GO GET ANOTHER CHARACTER
NXCD2:  .BLK    1           ; COUNTER TO LIMIT THE OCTAL STRING TO SIX DIGITS
S02B:   .BLK    1           ; "02B" RETURN ADDRESS
;
;*****************************************************************************
;       0 2 B   0 2 B   0 2 B   0 2 B   0 2 B   0 2 B   0 2 B   0 2 B   0 2 B
;*****************************************************************************
;*****************************************************************************
;       P O P T   P O P T   P O P T   P O P T   P O P T   P O P T   P O P T
;*****************************************************************************
;
; "POPT" POPS A WORD OFF THE TOS STACK
;
;       ENTRY:      POPT + 1
;       INPUT:      NONE
;       OUTPUT:     WORD POPPED OFF THE TOS STACK, IN POPT
;       REQUIRED:   NONE
;
POPT:   .BLK    1           ; ON EXIT, WORD POPPED OFF THE TOS STACK
        STA     3,SPOPT     ; SAVE RETURN ADDRESS
        LDA     2,.TOS      ; AC2 <-- STARTING ADDRESS OF TOS STACK
        LDA     1,TOSP      ; AC1 <-- TOS STACK POINTER
        ADD     1,2         ; ADD THESE TWO TO OFFSET TO THE TOP OF THE STACK
        LDA     0,0,2       ;
        STA     0,POPT      ; POPT <-- WORD ON TOP OF THE TOS STACK
        LDA     0,POS1      ;
        SUB     0,1         ;
        STA     1,TOSP      ; TOSP <-- TOSP - 1
        JMP     @SPOPT      ; RETURN
SPOPT:  .BLK    1           ; "POPT" RETURN ADDRESS
;
;*****************************************************************************
```

```
;       P O P T   P O P T   P O P T   P O P T   P O P T   P O P T   P O P T
;*****************************************************************************
;*****************************************************************************
;       P S B   P S B   P S B   P S B   P S B   P S B   P S B   P S B   P S B
;*****************************************************************************
;
; "PSB" BUILDS A REVERSE POLISH STACK IN IPS OF THE ASSERTION IN ATXT
;
;       ENTRY:      PSB + 1
;       INPUT:      NONE
;       OUTPUT:     STACK VALIDITY FLAG(0=INVALID,1=VALID), IN PSB
;       REQUIRED:   "GAT", "NTKN", "POPT", "PSHI", "PSHT", "PUTB", "PUTC"
;
PSB:    .BLK    1           ; ON EXIT, STACK VALIDITY FLAG(0=INVALID,1=VALID)
        STA     3,SPSB      ; SAVE RETURN ADDRESS
        LDA     0,ZERO      ;
        STA     0,PSB       ; ASSUME INVALID(PSB <-- 0)
        LDA     0,SPACE     ; OUTPUT A FENCE("$") AS THE ASSERTION INPUT PROMPT
        JSR     @.PUTC      ;
        LDA     0,DSIGN     ;
        JSR     @.PUTC      ;
        LDA     0,PCS5      ; MAKE NOTE OF THIS FENCE AS THE CURRENT(FIRST)
                            ; TOKEN(NTOK <-- 5)
        STA     0,NTOK      ;
        JSR     @.GAT       ; CALL "GAT" TO INPUT THE ASSERTION TEXT INTO ATXT
        LDA     0,NEG1      ; MOVE THE ASSERTION TEXT POINTER BACK TO
                            ; THE FRONT OF THE BUFFER(ATXTP <-- -1)
        STA     0,ATXTP     ;
        STA     0,IPSP      ; MARK THE INTERMEDIATE(IOS) AND OUTPUT(IPS) STACKS
                            ; EMPTY(TOSP <-- -1 AND IPSP <-- -1, RESPECTIVELY)
        STA     0,TOSP      ;
        LDA     2,.PSHT     ; PUSH A FENCE ONTO TOS(-1,0)
        LDA     0,NEG1      ;
        STA     0,0,2       ;
        JSR     1,2         ;
```

```
        LDA     2,.PSHT     ;
        LDA     0,ZERO      ;
        STA     0,0,2       ;
        JSR     1,2         ;
        LDA     2,.PSHI     ; PUSH A FENCE ONTO IPS(-1,0)
        LDA     0,NEG1      ;
        STA     0,0,2       ;
        JSR     1,2         ;
        LDA     2,.PSHI     ;
        LDA     0,ZERO      ;
        STA     0,0,2       ;
        JSR     1,2         ;
PSB1:   LDA     0,NTOK      ;
        STA     0,PTOK      ; PTOK <-- NTOK
        LDA     2,.NTKN     ; CALL "NTKN" TO FIND THE NEXT TOKEN IN THE ASSERTION
        JSR     3,2         ;
        LDA     0,@.NTKN    ; TOKEN VALIDITY FLAG
        STA     0,NTOK      ; NTOK <-- "NTKN" TOKEN CLASS
        MOV#    0,0,SNR     ; CHECK IF THE TOKEN WAS VALID(NTOK > 0)
        JMP     PSB2        ; NO -- INVALID TOKEN
        LDA     0,PTOK      ; YES -- LOOK UP THE (PTOK,NTOK) PAIR IN THE ALLOWED
                            ; TOKEN PAIR TABLE -- THE ATPT(5 X 5) IS STORED IN
                            ; ROW MAJOR ORDER, SO THE (PTOK,NTOK) PAIR IS OFFSET
                            ; FROM THE BEGINNING OF THE TABLE BY
                            ; 5 * (PTOK - 1) + (NTOK - 1),
                            ; OR (5 * PTOK) + NTOK - 6 ENTRIES
        MOV     0,1         ;
        ADDZL   0,0         ;
        ADD     1,0         ; AC0 <-- (5 * PTOK)
        LDA     1,NTOK      ;
        ADD     1,0         ; AC0 <-- (5 * PTOK) + NTOK
        LDA     1,POS6      ;
        SUB     1,0         ; AC0 <-- (5 * PTOK) + NTOK - 6
        LDA     2,.ATPT     ; AC2 <-- STARTING ADDRESS OF ATPT
        ADD     0,2         ; AC2 <-- ADDRESS OF (PTOK,NTOK) ENTRY
```

```
        LDA     0,0,2       ;
        MOV#    0,0,SNR     ; CHECK THE (PTOK,NTOK) ENTRY
        JMP     PSE2        ; INVALID PAIR
        LDA     0,NTOK      ; VALID PAIR -- CALCULATE THE DISPLACEMENT INTO THE
                            ; TOKEN JUMP TABLE -- .TJT + (NTOK - 1) -- AND BRANCH
                            ; TO THE APPROPRIATE TOKEN HANDLER
        LDA     1,POS1      ;
        SUB     1,0         ;
        LDA     2,.TJT      ;
        ADD     0,2         ;
        JMP     0,2         ;
PSE2:   LDA     2,.PUTB     ; FLAG THE INVALID TOKEN OR TOKEN PAIR -- DISPLAY
                            ; THE TWO CHARACTERS BEFORE AND AFTER THE LAST
                            ; CHARACTER PROCESSED IN ATXT TO HELP PINPOINT THE
                            ; ERROR -- AND RETURN
        LDA     0,.ERR4     ;
        STA     0,0,2       ;
        JSR     1,2         ;
        LDA     0,ATXTP     ;
        LDA     1,POS2      ;
        SUB     1,0         ;
        LDA     2,.ATXT     ;
        ADD     0,2         ;
        LDA     0,0,2       ;
        JSR     @.PUTC      ;
        LDA     0,1,2       ;
        JSR     @.PUTC      ;
        LDA     0,2,2       ;
        JSR     @.PUTC      ;
        LDA     0,3,2       ;
        JSR     @.PUTC      ;
        LDA     0,4,2       ;
        JSR     @.PUTC      ;
        LDA     0,QUOTE     ;
        JSR     @.PUTC      ;
```

```
        JMP     @SPSB       ;
ERR3:   LDA     2,.PUTB     ; FLAG UNMATCHED PARENTHESES AND RETURN
        LDA     0,.ERR5     ;
        STA     0,0,2       ;
        JSR     1,2         ;
        JMP     @SPSB       ;
TTCK:   .BLK    1           ; TOKEN CLASS OF CURRENT TOKEN
OPTD:   .BLK    1           ; OPERATOR ID SAVE LOCATION
OPER:   .BLK    1           ; OPERATOR PRIORITY SAVE LOCATION
PTCK:   .BLK    1           ; TOKEN CLASS OF PREVIOUS TOKEN
SPSB:   .BLK    1           ; "PSB" RETURN ADDRESS
.PSB1:  PSB1                ; ADDRESS OF "PSB1", ENTRY POINT IN "PSB" FOR
                            ;                    PROCESSING THE NEXT ASSERTION TOKEN
.TJT:   TJT                 ; STARTING ADDRESS OF TOKEN JUMP TABLE
TJT:    JMP     TOK1        ; TOKEN CLASS = 1 -- "("
        JMP     TOK2        ; TOKEN CLASS = 2 -- ")"
        JMP     TOK3        ; TOKEN CLASS = 3 -- OPERATOR
        JMP     TOK4        ; TOKEN CLASS = 4 -- OPERAND
        JMP     TOK5        ; TOKEN CLASS = 5 -- FENCE
;
;       TOKEN CLASS = 1 -- "(" -- PUSH IT ONTO TOS(10,0) IMMEDIATELY AND
;       GO GET THE NEXT TOKEN
;
TOK1:   LDA     2,.PSHT     ;
        LDA     0,POS10     ;
        STA     0,0,2       ;
        JSR     1,2         ;
        LDA     2,.PSHT     ;
        LDA     0,ZERO      ;
        STA     0,0,2       ;
        JSR     1,2         ;
        JMP     @.PSB1      ;
;
;       TOKEN CLASS = 2 -- ")" -- POP OPERATORS OFF TOS AND PUSH THEM ONTO IPS
;       UNTIL A MATCHING "(" IS FOUND -- DISCARD BOTH PARENTHESES AND
```

```
;       GO GET THE NEXT TOKEN
;
TOK2:   LDA     2,.POPT     ;
        JSR     1,2         ; POP PRIORITY OF TOP TOS OPERATOR
        LDA     2,.POPT     ;
        JSR     1,2         ; POP ID OF TOP TOS OPERATOR
        LDA     0,6,.POPT   ;
        STA     0,OPID      ; OPID <-- ID OF OPERATOR JUST POPPED OFF TOS
        LDA     1,NEG1      ;
        SUB#    0,1,SNR     ; CHECK IF THE OPERATOR JUST POPPED OFF TOS
                            ; WAS A FENCE(OPERATOR ID = -1)
        JMP     PSB3        ; YES -- UNMATCHED PARENTHESES
        LDA     1,POS10     ;
        SUB#    0,1,SNR     ; NO -- CHECK IF IT WAS A "("(OPERATOR ID = 10)
        JMP     @.PSB1      ; YES -- FOUND THE MATCHING ")" -- IGNORE THEM
                            ; AND GO GET THE NEXT TOKEN
        LDA     2,.PSHI     ; NOT A FENCE OR "(", SO OUTPUT THIS OPERATOR
                            ; ONTO IPS(1,OPID) AND GO TRY THE NEW TOP OF TOS
        LDA     0,POS1      ;
        STA     0,0,2       ;
        JSR     1,2         ;
        LDA     2,.PSHI     ;
        LDA     0,OPID      ;
        STA     0,0,2       ;
        JSR     1,2         ;
        JMP     TOK2        ;
;
;       TOKEN CLASS = 3 -- OPERATOR -- IF THE CURRENT OPERATOR'S PRIORITY IS
;       HIGHER THAN THAT OF THE OPERATOR ON THE TOP OF TOS, PUSH IT ONTO TOS
;       IMMEDIATELY ... OTHERWISE, POP OPERATORS OFF TOS AND PUSH THEM ONTO IPS
;       UNTIL THIS CONDITION IS MET, THEN PUSH IT ONTO TOS ... IN EITHER CASE,
;       GO GET THE NEXT TOKEN AFTER PUSHING THE CURRENT OPERATOR ONTO TOS
;
TOK3:   LDA     2,.NTKN     ;
        LDA     0,1,2       ;
```

```
        STA     0,OPID      ; OPID <-- CURRENT OPERATOR'S ID
        LDA     0,2,2       ;
        STA     0,OPPR      ; OPPR <-- CURRENT OPERATOR'S PRIORITY
TOK31:  LDA     2,.TOS      ;
        LDA     0,TOSP      ;
        ADD     0,2         ; AC2 <-- ADDRESS OF TOP OF TOS
        LDA     0,0,2       ; AC0 <-- PRIORITY OF OPERATOR ON TOP OF TOS
        LDA     1,OPPR      ; AC1 <-- CURRENT OPERATOR'S PRIORITY
        ADCZ#   0,1,SZC     ; SKIP IF AC1(CURRENT OPERATOR'S PRIORITY) <=
                            ; AC0(PRIORITY OF OPERATOR ON TOP OF TOS)
        JMP     TOK32       ; HIGHER PRIORITY THAN TOP OF STACK -- STACK IT
        LDA     2,.POPT     ; EQUAL OR LOWER PRIORITY THAN TOP OF STACK -- POP THE
                            ; TOS OPERATOR AND PUSH A1 ONTO
                            ; IPS(1,CURRENT TOS OPERATOR'S ID)
        JSR     1,2         ;
        LDA     2,.POPT     ;
        JSR     1,2         ;
        LDA     2,.PSHI     ;
        LDA     0,POS1      ;
        STA     0,0,2       ;
        JSR     1,2         ;
        LDA     2,.PSHI     ;
        LDA     0,@.POPT    ;
        STA     0,0,2       ;
        JSR     1,2         ;
        JMP     TOK31       ; GO TRY THE NEW TOP OF TOS
TOK32:  LDA     2,.PSHT     ; PUSH THE CURRENT OPERATOR ONTO TOS(OPID,OPPR) AND
                            ; GO GET THE NEXT TOKEN
        LDA     0,OPID      ;
        STA     0,0,2       ;
        JSR     1,2         ;
        LDA     2,.PSHT     ;
        LDA     0,OPPR      ;
        STA     0,0,2       ;
        JSR     1,2         ;
```

```
        JMP     @.PSB1      ;
;
;       TOKEN CLASS = 4 -- OPERAND -- PUSH IT ONTO IPS IMMEDIATELY AND
;       GO GET THE NEXT TOKEN
;
TOK4:   LDA     2,.NTKN     ;
        LDA     0,1,2       ;
        LDA     2,.PSH1     ;
        STA     0,0,2       ;
        JSR     1,2         ; PUSH THE OPERAND TYPE ONTO IPS
        LDA     2,.NTKN     ;
        LDA     0,2,2       ;
        LDA     2,.PSH1     ;
        STA     0,0,2       ;
        JSR     1,2         ; PUSH THE OPERAND VALUE ONTO IPS
        JMP     @.PSB1      ;
;
;       TOKEN CLASS = 5 -- FENCE -- POP ENTRIES OFF TOS AND PUSH THEM ONTO IPS
;       UNTIL THE MATCHING FENCE IS FOUND, THEN RETURN
;
TOK5:   LDA     2,.POPT     ;
        JSR     1,2         ; POP PRIORITY OF TOP TOS OPERATOR
        LDA     2,.POPT     ;
        JSR     1,2         ; POP ID OF TOP TOS OPERATOR
        LDA     0,@.POPT    ;
        STA     0,OPID      ; OPID <-- ID OF OPERATOR JUST POPPED OFF TOS
        LDA     1,POS10     ;
        SUB#    0,1,SNR     ; CHECK IF OPERATOR JUST POPPED OFF TOS
                            ; WAS A "("(OPERATOR ID = 10)
        JMP     PSB3        ; YES -- UNMATCHED PARENTHESES
        LDA     1,NEG1      ;
        SUB#    0,1,SNR     ; NO -- CHECK IF IT WAS A FENCE(OPERATOR ID = -1)
        JMP     TOK51       ; YES -- GO PUSH A CLOSING FENCE ON IPS AND RETURN
        LDA     2,.PSH1     ; NO -- PUSH THIS OPERATOR ONTO IPS(1,OPID)
        LDA     0,POS1      ;
```

```
        STA     0,0,2       ;
        JSR     1,2         ;
        LDA     2,.PSHI     ;
        LDA     0,CPID      ;
        STA     0,0,2       ;
        JSR     1,2         ;
        JMP     TOK5        ; GO TRY THE NEW TOP OF TOS
TOK51:  LDA     2,.PSHI     ; ALL DONE -- PUSH A FENCE ONTO IPS(-1,0), MARK THE
                            ; STACK VALID(PSB <-- 1) AND RETURN
        LDA     0,NEG1      ;
        STA     0,0,2       ;
        JSR     1,2         ;
        LDA     2,.PSHI     ;
        LDA     0,ZERO      ;
        STA     0,0,2       ;
        JSR     1,2         ;
        LDA     0,POS1      ;
        STA     0,@.PSB     ;
        JMP     @SPSB       ;
;
;*****************************************************************************
;       P S B   P S B   P S B   P S B   P S B   P S B   P S B   P S B   P S B   P S B
;*****************************************************************************
;*****************************************************************************
;       P S E   P S E   P S E   P S E   P S E   P S E   P S E   P S E   P S E   P S E
;*****************************************************************************
;
; "PSE" EVALUATES A REVERSE POLISH STACK BUILT BY "PSB"
;
;       ENTRY:      PSE + 2
;       INPUT:      ADDRESS OF STACK TO BE EVALUATED, IN PSE
;       OUTPUT:     LOGICAL EVALUATION OF STACK(0=FLASE,1=TRUE),
;                   IN PVAL(PSE + 1)
;       REQUIRED:   NONE
;
```

```
PSE:    .BLK    1           ; ON ENTRY, ADDRESS OF STACK TO BE EVALUATED
VAL:    .BLK    1           ; ON EXIT, LOGICAL EVALUATION OF STACK(0=FALSE,1=TRUE)
        STA     3,SPSE      ; SAVE RETURN ADDRESS
        LDA     2,PSE       ; COPY STACK TO BE EVALUATED TO SCRATCHPAD POLISH STACK
        LDA     3,.SPS      ;
        LDA     0,0,2       ;
        STA     0,0,3       ;
        LDA     1,1,2       ;
        STA     1,1,3       ;
PSE1:   INC     2,2         ;
        INC     2,2         ;
        INC     3,3         ;
        INC     3,3         ;
        LDA     0,0,2       ;
        STA     0,0,3       ;
        LDA     1,1,2       ;
        STA     1,1,3       ;
        MOVL#   0,0,SNC     ; CHECK FOR A FENCE
        JMP     PSE1        ; NO -- MOVE THE NEXT ENTRY
        LDA     0,ZERO      ; YES -- MOVE THE SCRATCHPAD POLISH STACK POINTER
                            ; BACK TO THE BEGINNING OF THE STACK(SPSP <-- 0)
        STA     0,SPSP      ;
PSE2:   LDA     0,SPSP      ;
        INC     0,0         ;
        INC     0,0         ;
        STA     0,SPSP      ; LOOK AT THE NEXT STACK ENTRY(SPSP <-- SPSP + 2)
        LDA     2,.SPS      ;
        ADD     0,2         ; AC2 <-- ADDRESS OF NEXT STACK ENTRY
        LDA     0,0,2       ; AC0 <-- ID FIELD OF NEXT STACK ENTRY
        MOVL#   0,0,SNC     ; CHECK IF IT IS A FENCE(ID = -1)
        JMP     PSE4        ; NO -- OPERATOR, LITERAL OPERAND OR SYMBOLIC OPERAND
        LDA     0,SPSP      ; YES -- THE STACK'S VALUE IS IN THE VALUE FIELD OF
                            ; THE ONLY STACK ENTRY NOT MARKED UNUSED(ID = 0)
        LDA     2,.SPS      ;
        ADD     0,2         ; AC2 <-- ADDRESS OF FENCE
```

```
        LDA     1,POS1      ;
PSE3:   SUB     1,2         ; BACK UP TO THE PREVIOUS ENTRY(AC2 <-- AC2 - 2)
        SUB     1,2         ;
        LDA     0,0,2       ;
        MOV#    0,0,SNR     ; CHECK IF IT IS MARKED UNUSED(ID = 0)
        JMP     PSE3        ; YES -- BACK UP AND TRY AGAIN
        LDA     0,1,2       ; NO -- THIS ENTRY'S VALUE WORD IS THE STACK'S
                            ; OVERALL VALUE
        MOV#    0,0,SZR     ; MAKE SURE THE STACK VALUE IS 000000/000001
        LDA     0,LOGT      ; NON-ZERO, SO MAKE IT IDENTICALLY 000001
        STA     0,PVAL      ; PVAL <-- STACK VALUE AND RETURN
        JMP     @SPSE       ;
PSE4:   MOVR    0,0,SNC     ; CHECK IF THE CURRENT ENTRY IS AN OPERATOR(ID = 1)
        JMP     PSE7        ; NO -- LITERAL OPERAND OR SYMBOLIC OPERAND
        LDA     0,SPSP      ; YES -- CALCULATE THE DISPLACEMENT INTO THE OPERATOR
                            ; JUMP TABLE -- .OJT + ENCODED OPERATOR -- TO
                            ; BRANCH TO THE APPROPRIATE HANDLER
        LDA     2,.SPS      ;
        ADD     0,2         ;
        LDA     0,1,2       ; AC0 <-- ENCODED OPERATOR
        LDA     1,.OJT      ; AC1 <-- STARTING ADDRESS OF OPERATOR JUMP TABLE
        ADD     0,1         ;
        STA     1,OJTA      ; OJTA <-- OPERATOR JUMP TABLE ADDRESS
        LDA     1,POS1      ; NOW, BACK UP IN THE STACK TO FIND THE
                            ; OPERATOR'S OPERANDS
PSE5:   SUB     1,2         ; BACK UP TO THE PREVIOUS ENTRY(AC2 <-- AC2 - 2)
        SUB     1,2         ;
        LDA     0,0,2       ; AC0 <-- PREVIOUS ENTRY'S ID FIELD
        MOV#    0,0,SNR     ; CHECK IF IT IS MARKED USED(ID > 0)
        JMP     PSE5        ; NO -- BACK UP AND TRY AGAIN
        LDA     0,1,2       ;
        STA     0,OPRB      ; YES -- FIRST OPERAND FOUND IS OPRB
        LDA     0,ZERO      ;
        STA     0,0,2       ; MARK THIS ENTRY UNUSED NOW(ID <-- 0)
PSE6:   SUB     1,2         ; BACK UP TO THE PREVIOUS ENTRY(AC2 <-- AC2 - 2)
```

```
        SUB     1,2         ;
        LDA     0,0,2       ; AC0 <-- PREVIOUS ENTRY'S ID FIELD
        MOV#    0,0,SNR     ; CHECK IF IT IS MARKED USED(ID > 0)
        JMP     PSE6        ; NO -- BACK UP AND TRY AGAIN
        LDA     0,1,2       ;
        STA     0,OPRA      ; YES -- SECOND OPERAND FOUND IS OPRA
        LDA     0,ZERO      ;
        STA     0,0,2       ; MARK THIS ENTRY UNUSED NOW(ID <-- 0)
        LDA     0,OPRB      ; AC0 <-- OPERAND B
        LDA     1,OPRA      ; AC1 <-- OPERAND A
        JMP     @OJTA       ; JUMP TO THE APPROPRIATE OPERAND HANDLER
PSE7:   MOVR    0,0,SNC     ; CHECK IF IT IS A SYMBOLIC OPERAND(ID = 2)
        JMP     PSE2        ; NO -- MUST BE A LITERAL OPERAND -- IGNORE IT
        LDA     0,SPSP      ; YES -- ITS VALUE FIELD IS THE ADDRESS OF ITS VALUE
        LDA     2,.SPS      ;
        ADD     0,2         ;
        LDA     0,@1,2      ;
        STA     0,1,2       ; VALUE FIELD <-- (VALUE FIELD)
        JMP     PSE2        ; GO PROCESS THE NEXT STACK ENTRY
OJTA:   .BLK    1           ; OPERATOR JUMP TABLE ADDRESS
OPRA:   .BLK    1           ; LEFT OPERAND IN POLISH TRIPLES
OPRB:   .BLK    1           ; RIGHT OPERAND IN POLISH TRIPLES
SPSP:   .BLK    1           ; "PSE" RETURN ADDRESS
.OJT:   OJT                 ; ADDRESS OF OPERATOR JUMP TABLE
OJT:    JMP     OP0         ; OPERATOR 0 -- ADDITION
        JMP     OP1         ; OPERATOR 1 -- SUBTRACTION
        JMP     OP2         ; OPERATOR 2 -- EQUAL
        JMP     OP3         ; OPERATOR 3 -- NOT-EQUAL
        JMP     OP4         ; OPERATOR 4 -- LESS-THAN
        JMP     OP5         ; OPERATOR 5 -- GREATER-THAN
        JMP     OP6         ; OPERATOR 6 -- LESS-THAN-OR-EQUAL-TO
        JMP     OP7         ; OPERATOR 7 -- GREATER-THAN-OR-EQUAL-TO
        JMP     OP8         ; OPERATOR 8 -- AND
        JMP     OP9         ; OPERATOR 9 -- OR
OP0:    ADD     0,1         ; ADD OPERATOR -- AC1 + AC0
```

```
        MOV     1,2         ;
        JMP     OPEND       ; AC2 <-- OPRA + OPRB
OPR1:   SUB     0,1         ; SUBTRACT OPERATOR -- AC1 - AC0
        MOV     1,2         ;
        JMP     OPEND       ; AC2 <-- OPRA - OPRB
OPR2:   LDA     2,LOGT      ; EQUAL OPERATOR -- ASSUME TRUE
        SUB#    0,1,SZR     ; SKIP IF AC1 = AC0
        LDA     2,LOGF      ; FALSE
        JMP     OPEND       ; AC2 <-- OPRA = OPRB
OPR3:   LDA     2,LOGT      ; NOT-EQUAL OPERATOR -- ASSUME TRUE
        SUB#    0,1,SNR     ; SKIP IF AC1 <> AC0
        LDA     2,LOGF      ; FALSE
        JMP     OPEND       ; AC2 <-- OPRA <> OPRB
OPR4:   LDA     2,LOGT      ; LESS-THAN OPERATOR -- ASSUME TRUE
        SUBL#   0,1,SNC     ; SKIP IF AC1 < AC0
        LDA     2,LOGF      ; FALSE
        JMP     OPEND       ; AC2 <-- OPRA < OPRB
OPR5:   LDA     2,LOGT      ; GREATER-THAN OPERATOR -- ASSUME TRUE
        SUBL#   1,0,SNC     ; SKIP IF AC1 > AC0
        LDA     2,LOGF      ; FALSE
        JMP     OPEND       ; AC2 <-- OPRA > OPRB
OPR6:   LDA     2,LOGT      ; LESS-THAN-OR-EQUAL-TO OPERATOR -- ASSUME TRUE
        SUBL#   1,0,SZC     ; SKIP IF AC1 <= AC0
        LDA     2,LOGF      ; FALSE
        JMP     OPEND       ; AC2 <-- OPRA <= OPRB
OPR7:   LDA     2,LOGT      ; GREATER-THAN-OR-EQUAL-TO OPERATOR -- ASSUME TRUE
        SUBL#   0,1,SZC     ; SKIP IF AC1 >= AC0
        LDA     2,LOGF      ; FALSE
        JMP     OPEND       ; AC2 <-- OPRA >= OPRB
OPR8:   AND     0,1         ; AND OPERATOR -- AC1 & AC0
        MOV     1,2         ;
        JMP     OPEND       ; AC2 <-- OPRA & OPRB
OPR9:   COM     0,0         ; OR OPERATOR -- AC1 : AC0
        AND     0,1         ;
        ADC     0,1         ;
```

```
        MOV     1,2             ; AC2 <-- OPRA : OPRB
OPELE:  LDA     0,SPSP          ;
        LDA     3,.SPS          ;
        ADD     0,3             ;
        STA     2,1,3           ; PUT THE RESULT INTO THE VALUE FIELD OF THE
                                ; OPERATOR STACK ENTRY JUST PROCESSED AND GO PROCESS
                                ; THE NEXT STACK ENTRY
        JMP     PSE2            ;
;
;*****************************************************************************
;       P S E   P S E   P S E   P S E   P S E   P S E   P S E   P S E   P S E
;*****************************************************************************
;*****************************************************************************
;       P S H I   P S H I   P S H I   P S H I   P S H I   P S H I   P S H I
;*****************************************************************************
;
; "PSHI" PUSHES A WORD ONTO THE IPS STACK
;
;       ENTRY:      PSHI + 1
;       INPUT:      WORD TO BE PUSHED ONTO THE IPS STACK, IN PSHI
;       OUTPUT:     NONE
;       REQUIRED:   "PUTB"
;
PSHI:   .BLK    1               ; ON ENTRY, WORD TO BE PUSHED ONTO THE IPS STACK
        STA     3,SPSH1         ; SAVE RETURN ADDRESS
        LDA     0,IPSP          ;
        INC     0,0             ;
        STA     0,IPSP          ; IPSP <-- IPSP + 1
        LDA     1,IPSM          ; AC1 <-- IPSM(MAXIMUM VALUE OF IPSP)
        ADCZ#   1,0,SNC         ; SKIP IF AC0(IPSP + 1) > AC1(IPSM)
        JMP     PSHI1           ;
        LDA     2,.PUTB         ; NO MORE ROOM -- OUTPUT A FATAL ERROR
                                ; MESSAGE(IPS OVERFLOW) AND RETURN TO "ALA" DIRECTLY
        LDA     0,.ERR3         ;
        STA     0,0,2           ;
```

```
        JSR     1,2             ;
        JMP     @.ALA1          ;
PSHI1:  LDA     2,.IPS          ; THERE WAS ROOM, SO ADD THE CHARACTER AND RETURN
        ADD     0,2             ;
        LDA     0,PSHI          ;
        STA     0,0,2           ;
        JMP     @SPSHI          ;
SPSHT:  .BLK    1               ; "PSHI" RETURN ADDRESS
;
;*****************************************************************************
;       P S H I   P S H I   P S H I   P S H I   P S H I   P S H I   P S H I
;*****************************************************************************
;*****************************************************************************
;       P S H T   P S H T   P S H T   P S H T   P S H T   P S H T   P S H T
;*****************************************************************************
;
; "PSHT" PUSHES A WORD ONTO THE TOS STACK
;
;       ENTRY:      PSHT + 1
;       INPUT:      WORD TO BE PUSHED ONTO THE TOS STACK, IN PSHT
;       OUTPUT:     NONE
;       REQUIRED:   "PUTB"
;
PSHT:   .BLK    1               ; ON ENTRY, WORD TO BE PUSHED ONTO THE TOS STACK
        STA     3,SPSHT         ; SAVE RETURN ADDRESS
        LDA     0,TOSP          ;
        INC     0,0             ;
        STA     0,TOSP          ; TOSP <-- TOSP + 1
        LDA     1,TOSM          ; AC1 <-- TOSM(MAXIMUM VALUE OF TOSP)
        ADCZ#   1,0,SNC         ; SKIP IF AC0(TOSP + 1) > AC1(TOSM)
        JMP     PSHT1           ;
        LDA     2,.PUTB         ; NO MORE ROOM -- OUTPUT A FATAL ERROR
                                ; MESSAGE(TOS OVERFLOW) AND RETURN TO "ALA" DIRECTLY
        LDA     0,.ERR2         ;
        STA     0,0,2           ;
```

```
        JSR     1,2             ;
        JMP     @.ALA1          ;
PSHT1:  LDA     2,.TOS          ; THERE WAS ROOM, SO ADD THE CHARACTER AND RETURN
        ADD     0,2             ;
        LDA     0,PSHT          ;
        STA     0,0,2           ;
        JMP     @SPSHT          ;
SPSHT:  .BLK    1               ; "PSHT" RETURN ADDRESS
;
;*****************************************************************************
;       P S H T   P S H T   P S H T   P S H T   P S H T   P S H T   P S H T
;*****************************************************************************
;*****************************************************************************
;       P U T B   P U T B   P U T B   P U T B   P U T B   P U T B   P U T B
;*****************************************************************************
;
; "PUTB" PERFORMS BUFFERED TELETYPE OUTPUT
;
;       ENTRY:      PUTB + 1
;       INPUT:      STARTING ADDRESS OF TEXT BUFFER TO BE OUTPUT, IN PUTB
;       OUTPUT:     NONE
;       REQUIRED:   "PUTC"
;
PUTB:   .BLK    1               ; ON ENTRY, STARTING ADDRESS OF TEXT BUFFER
        STA     3,SPUTB         ; SAVE RETURN ADDRESS
        LDA     2,PUTB          ; AC2 <-- STARTING ADDRESS OF BUFFER
PUTB1:  LDA     0,0,2           ; AC0 <-- NEXT WORD OF BUFFER
        MOV#    0,0,SNR         ; CHECK FOR A NULL WORD
        JMP     @SPUTB          ; YES -- END OF THE TEXT
        JSR     @.PUTC          ; NO -- OUTPUT THE RIGHT CHARACTER
        MOVS    0,0             ; SWAP THE CHARACTERS
        JSR     @.PUTC          ; OUTPUT THE LEFT CHARACTER
        INC     2,2             ; INCREMENT AC2 TO POINT TO THE
                                ; NEXT WORD OF THE BUFFER AND GO BACK
        JMP     PUTB1           ;
```

```
PUTB:  .BLK   1           ; "PUTB" RETURN ADDRESS
;
;*****************************************************************************
;       P U T B   P U T B   P U T B   P U T B   P U T B   P U T B   P U T B
;*****************************************************************************
;*****************************************************************************
;       P U T C   P U T C   P U T C   P U T C   P U T C   P U T C   P U T C
;*****************************************************************************
;
; "PUTC" PERFORMS SINGLE CHARACTER TELETYPE OUTPUT
;
;       ENTRY:      PUTC
;       INPUT:      CHARACTER TO BE OUTPUT, IN AC0
;       OUTPUT:     NONE
;       REQUIRED:   NONE
;
PUTC:  SKPBZ  TTO         ; WAIT UNTIL THE TELETYPE IS NOT BUSY
       JMP    .-1         ;
       DOAS   0,TTO       ; OUTPUT THE CHARACTER IN AC0
       JMP    0,3         ; RETURN
;
;*****************************************************************************
;       P U T C   P U T C   P U T C   P U T C   P U T C   P U T C   P U T C
;*****************************************************************************
;*****************************************************************************
;       R N A C   R N A C   R N A C   R N A C   R N A C   R N A C   R N A C
;*****************************************************************************
;
; "RNAC" RETRIEVES THE CHARACTER IN ATXT, THE ASSERTION TEXT BUFFER,
;       FOLLOWING THE ONE CURRENTLY POINTED TO BY ATXTP, THE ASSERTION
;       TEXT POINTER
;
;       ENTRY:      RNAC
;       INPUT:      NONE
;       OUTPUT:     NEXT ASSERTION TEXT CHARACTER, IN AC0
```

```
;         REQUIRED:    NONE
;         NOTE:        ATXTP, THE ASSERTION TEXT POINTER, IS ADVANCED
;
RNAC:   LDA     2,.ATXT     ; AC2 <-- STARTING ADDRESS OF ASSERTION TEXT BUFFER
        LDA     1,ATXTP     ; AC1 <-- ASSERTION TEXT POINTER
        INC     1,1         ; INCREMENT THE ASSERTION TEXT POINTER
        STA     1,ATXTP     ; ATXTP <-- ATXTP + 1
        ADD     1,2         ; ADD THESE TWO TO OFFSET TO THE NEXT CHARACTER
        LDA     0,0,2       ; AC0 <-- NEXT ASSERTION TEXT CHARACTER
        JMP     0,3         ; RETURN
;
;*****************************************************************************
;         R N A C   R N A C   R N A C   R N A C   R N A C   R N A C   R N A C
;*****************************************************************************
;*****************************************************************************
;         S P 3   S P 3   S P 3   S P 3   S P 3   S P 3   S P 3   S P 3   S P 3
;*****************************************************************************
;
; "SP3" OUTPUTS THREE SPACES
;
;         ENTRY:       SP3
;         INPUT:       NONE
;         OUTPUT:      NONE
;         REQUIRED:    "PUTC"
;
SP3:    STA     3,SSP3      ; SAVE RETURN ADDRESS
        LDA     0,SPACE     ; AC0 <-- ASCII SPACE
        JSR     @.PUTC      ; ONE,
        JSR     @.PUTC      ;   TWO,
        JSR     @.PUTC      ;     THREE
        JMP     @SSP3       ; RETURN
SSP3:   .BLK    1           ; "SP3" RETURN ADDRESS
;
;*****************************************************************************
;         S P 3   S P 3   S P 3   S P 3   S P 3   S P 3   S P 3   S P 3   S P 3
```

```
;**********************************************************************
;**********************************************************************
;       S R C H   S R C H   S R C H   S R C H   S R C H   S R C H   S R C H
;**********************************************************************
;
; "SRCH" SEARCHES FOR AN IDENTIFIER IN THE SYMBOL TABLE
;
;       ENTRY:      SRCH + 1
;       INPUT:      ADDRESS OF IDENTIFIER TO BE SEARCHED FOR, IN SRCH
;       OUTPUT:     ABSOLUTE ADDRESS(-1 IF NOT FOUND) OF IDENTIFIER, IN SRCH
;       REQUIRED:   NONE
;
SRCH:   .BLK    1           ; ON ENTRY, ADDRESS OF IDENTIFIER TO BE SEARCHED FOR
        STA     3,SSRCH     ; SAVE RETURN ADDRESS
        LDA     0,PCS6      ;
        LDA     1,SYMNO     ;
        ADD     0,1         ;
        STA     1,NSRCH     ; NSRCH <-- SYMNO(NUMBER OF USER SYMBOLS) +
                            ; 6(NUMBER OF RESERVED SYMBOLS)
        LDA     0,.SYMB     ;
        STA     0,PSRCH     ; PSRCH <-- STARTING ADDRESS OF SYMBOL TABLE
SRCH1:  LDA     2,SRCH      ; COMPARE THE THREE WORDS(5 CHARACTERS PLUS <00>)
                            ; OF THE IDENTIFIER BEING SEARCHED FOR WITH THE THREE
                            ; WORDS OF THE NEXT SYMBOL TABLE IDENTIFIER --
                            ; THE IDENTIFIER HAS BEEN FOUND ONLY IF ALL THREE MATCH
        LDA     3,PSRCH     ;
        LDA     0,0,2       ;
        LDA     1,0,3       ;
        SUB#    0,1,SZR     ;
        JMP     SRCH2       ; MIS-MATCH ON WORD 1
        LDA     0,1,2       ;
        LDA     1,1,3       ;
        SUB#    0,1,SZR     ;
        JMP     SRCH2       ; MIS-MATCH ON WORD 2
        LDA     0,2,2       ;
```

```
        LDA     1,2,3       ;
        SUB#    0,1,SZR     ;
        JMP     SRCH2       ; MIS-MATCH ON WORD 3
        LDA     0,3,3       ; FOUND IT -- SRCH <-- IDENTIFIER'S ABSOLUTE ADDRESS
        STA     0,SRCH      ;
        JMP     @SSRCH      ;
SRCH2:  LDA     0,POS4      ;
        LDA     1,PSRCH     ;
        ADD     0,1         ;
        STA     1,PSRCH     ; PSRCH <-- PSRCH + 4 TO LOOK AT THE NEXT SYMBOL
                            ; TABLE ENTRY
        DSZ     NSRCH       ; CHECK FOR THE END OF THE TABLE
        JMP     SRCH1       ; NO -- KEEP LOOKING
        LDA     0,NEG1      ; YES -- MARK THE IDENTIFIER AS UNDEFINED AND RETURN
        STA     0,SRCH      ;
        JMP     @SSRCH      ;
NSRCH:  .BLK    1           ; NUMBER OF ENTRIES TO BE LOOKED AT
PSRCH:  .BLK    1           ; SEARCH POINTER
SSRCH:  .BLK    1           ; "SRCH" RETURN ADDRESS
;
;*****************************************************************************
;       S R C H   S R C H   S R C H   S R C H   S R C H   S R C H   S R C H
;*****************************************************************************
;*****************************************************************************
;       T I S R   T I S R   T I S R   T I S R   T I S R   T I S R   T I S R
;*****************************************************************************
;
;       "TISR" SERVICES TELETYPE INPUT INTERRUPTS DURING PROGRAM
;              EXECUTION IN "ECM"
;
;       ENTRY:      TISR
;       INPUT:      NONE
;       OUTPUT:     NONE
;       REQUIRED:   "DMP"
;
```

```
TTSR:   STA     3,STIA3     ; SAVE INTERRUPTED CPU STATE
        STA     2,STIA2     ;
        STA     1,STIA1     ;
        STA     0,STIA0     ;
        MOVR    0,0         ;
        STA     0,STICY     ;
        LDA     0,0         ; SAVE RETURN ADDRESS
        STA     0,STISR     ;
        DIAC    0,TTI       ; AC0 <-- INPUT CHARACTER AND CLEAR TELETYPE INPUT
        LDA     1,CMASK     ; AC1 <-- 000177(ASCII CHARACTER MASK)
        AND     1,0         ; MASK OFF THE PARITY BIT
        LDA     1,ESC       ;
        SUB#    0,1,SNR     ; CHECK IF IT WAS AN "ESC"
        JMP     TISR1       ; YES -- ABNORMALLY TERMINATE PROGRAM EXECUTION
        LDA     0,STICY     ; NO -- IGNORE IT -- RESTORE INTERRUPTED CPU STATE
        MOVL    0,0         ;
        LDA     0,STIA0     ;
        LDA     1,STIA1     ;
        LDA     2,STIA2     ;
        LDA     3,STIA3     ;
        NIOS    TTI         ; RESTART THE TELETYPE INPUT
        JMP     @STISR      ; RETURN TO THE POINT OF INTERRUPTION IN "ECM"
TTSR1:  LDA     2,..DMP     ; CALL "DMP" TO OUTPUT THE USER PROGRAM STATE
                            ; AT AN ABNORMAL TERMINATION(DMP <-- 4)
        LDA     0,POS4      ;
        STA     0,0,2       ;
        JSR     1,2         ;
        JMP     @.ALA1      ; RETURN DIRECTLY TO "ALA"
STIA0:  .BLK    1           ; TELETYPE INPUT INTERRUPT AC0
STIA1:  .BLK    1           ; TELETYPE INPUT INTERRUPT AC1
STIA2:  .BLK    1           ; TELETYPE INPUT INTERRUPT AC2
STIA3:  .BLK    1           ; TELETYPE INPUT INTERRUPT AC3
STICY:  .BLK    1           ; TELETYPE INPUT INTERRUPT CARRY BIT
STISR:  .BLK    1           ; "TISR" RETURN ADDRESS
;
```

```
;*****************************************************************************
;       T I S R   T I S R   I I S R   T I S R   I I S R   T I S P   T I S R
;*****************************************************************************
;*****************************************************************************
;       U R E L   B U F F E R S ,   S T A C K S   A N D   T A B L E S
;*****************************************************************************
ATBT:   1                       ; "(" -- "("            ALLOWED TOKEN PAIR TABLE
        1                       ; "(" -- ")"              (0=INVALID,1=VALID)
        0                       ; "(" -- OPERATOR
        1                       ; "(" -- OPERAND
        0                       ; "(" -- "$"
        0                       ; ")" -- "("
        1                       ; ")" -- ")"
        1                       ; ")" -- OPERATOR
        0                       ; ")" -- OPERAND
        1                       ; ")" -- "$"
        1                       ; OPERATOR -- "("
        0                       ; OPERATOR -- ")"
        0                       ; OPERATOR -- OPERATOR
        1                       ; OPERATOR -- OPERAND
        0                       ; OPERATOR -- "$"
        0                       ; OPERAND -- "("
        1                       ; OPERAND -- ")"
        1                       ; OPERAND -- OPERATOR
        0                       ; OPERAND -- OPERAND
        1                       ; OPERAND -- "$"
        1                       ; "$" -- "("
        0                       ; "$" -- ")"
        0                       ; "$" -- OPERATOR
        1                       ; "$" -- OPERAND
        0                       ; "$" -- "$"
ATXT:   .BLK    620             ; ASSERTION TEXT BUFFER
APS:    .BLK    100             ; BREAKPOINTING ASSERTION POLISH STACK
DFTC:   .BLK    12              ; DFT COUNT VECTOR
DFTD:   .BLK    12              ; DFT DESTINATION VECTOR
```

```
DFTS:   .BLK    12          ; DF1 SOURCE VECTOR
ERR1:   .TXT    #<137>INVALID CHARACTER#
ERR2:   .TXT    #<15><12><12>*****ERROR*****    TOS OVERFLOW #
ERR3:   .TXT    #<15><12><12>*****ERROR*****    IPS OVERFLOW #
ERR4:   .TXT    #<15><12>*****ERROR*****    INVALID TOKEN NEAR "#
ERR5:   .TXT    #<15><12>*****ERROR*****    UNMATCHED PARENTHESES #
ERR6:   .TXT    #<15><12><12>*****ERROR*****    ATX1 OVERFLOW#
ERR7:   .TXT    #<137>INVALID ACCUMULATOR#
ERR8:   .TXT    #<137>INVALID COMMAND#
IPS:    .BLK    100         ; IMMEDIATE ASSERTION POLISH STACK
OUT1:   .TXT    #<15><15><12><12>? #
OUT2:   .TXT    #<15><12>       DYNAMIC FLOW TRACE <15><12>#
OUT3:   .TXT    #<15><12>   #
OUT4:   .TXT    # ---> #
OUT5:   .TXT    # (#
OUT6:   .TXT    #) #
        .TXTN   1
OUT7:   .TXT    #<15><12>                ADDR      INST     CARRY     AC-0 #
        .TXT    #     AC-1      AC-2      AC-3#
        .TXT    #<15><12>              ------    ------    -----    ------#
        .TXTM   0
        .TXT    #    ------    ------    ------ #
OUT8:   .TXT    #<15><12>HALT:      #
OUT9:   .TXT    #<15><12>FAILURE:   #
OUTA:   .TXT    #<15><12>PREVIOUS: #
OUTB:   .TXT    # AC-#
OUTC:   .TXT    #<15><12>    FALSE#
OUTD:   .TXT    #<15><12>    TRUE #
        .TXTN   1
OUTE:   .TXT    #<15><12>          SYMBOL TABLE#
        .TXT    #<15><12><12>LOCATION      SYMBOL      ADDRESS#
        .TXTN   0
        .TXT    #<15><12>--------      ------      ------- #
OUTF:   .TXT    #<15><15><12> #
OUTG:   .TXT    #<15><12>ESCAPE:    #
```

```
SPS:    .BLK    100         ; SCRATCHPAD POLISH STACK
        .TXTM   1
        .TXTN   1
SYMP:   .TXT    #*AC0<00><00>#                  ; RESERVED PORTION OF SYMBOL TABLE
        NEWA0
        .TXT    #*AC1<00><00>#
        NEWA1
        .TXT    #*AC2<00><00>#
        NEWA2
        .TXT    #*AC3<00><00>#
        NEWA3
        .TXT    #*CBIT<00>#
        NEWCY
        .TXT    #*PC<00><00><00>#
        NEWPC
        .TXTM   0
        .TXTN   0
SYMT:   .BLK    400         ; USER-DEFINED PORTION OF SYMBOL TABLE
OPS:    .BLK    100         ; TEMPORARY OPERAND STACK
;*****************************************************************
;       N R E L   B U F F E R S ,   S T A C K S   A N D   T A B L E S
;*****************************************************************
;
;*****************************************************************
;*****************************************************************
;       A L A D D I N   D E B U G G E R  --  V E R S I O N   1
;*****************************************************************
;*****************************************************************
        .END
```

APPENDIX E: CALL TREES OF THE ALADDIN ROUTINES

The figures in this appendix show the first-level subroutine call trees for the command monitor and command handlers, and the complete subroutine call trees for the utility routines (no trees are shown for primitive utilities such as "GETC" and "PUTC" which call no other routines). At all levels in the trees, those routines called directly by some other routine are listed alphabetically from left to right below the calling routine.

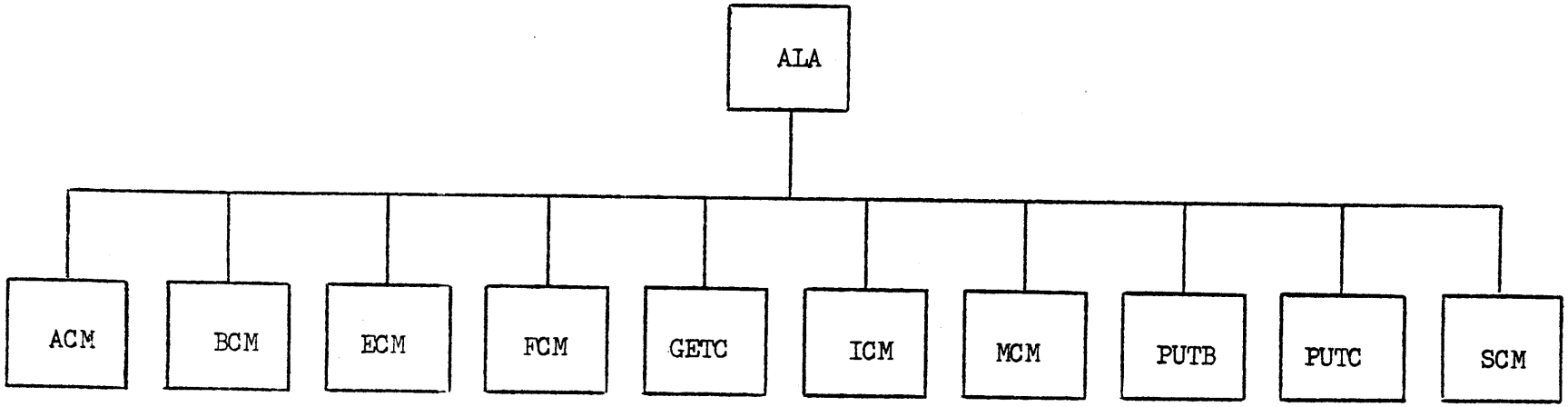


Fig. A4. Call tree for the command monitor "ALA".

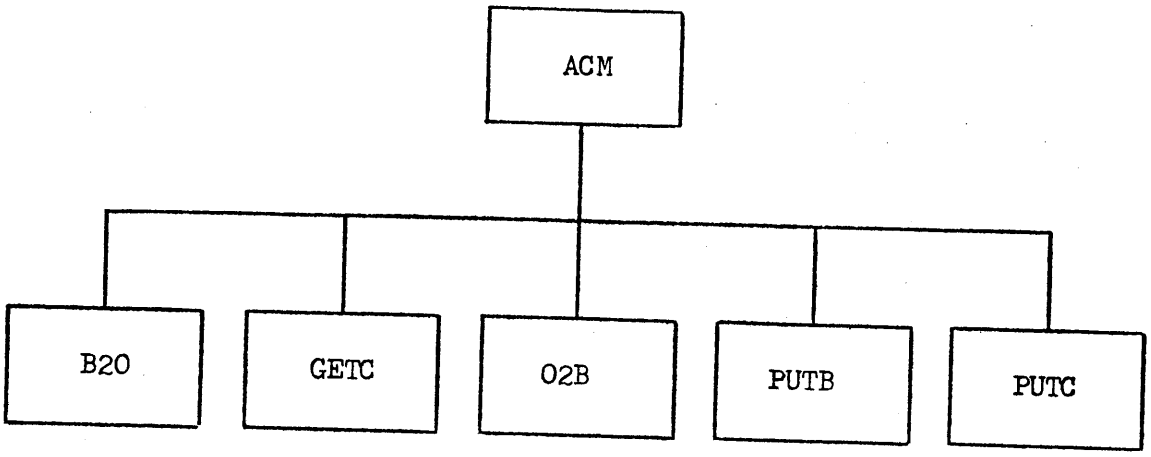


Fig. A5. Call tree for the command handler "ACM".

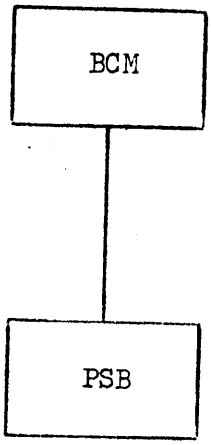


Fig. A6. Call tree for the command handler "BCM".

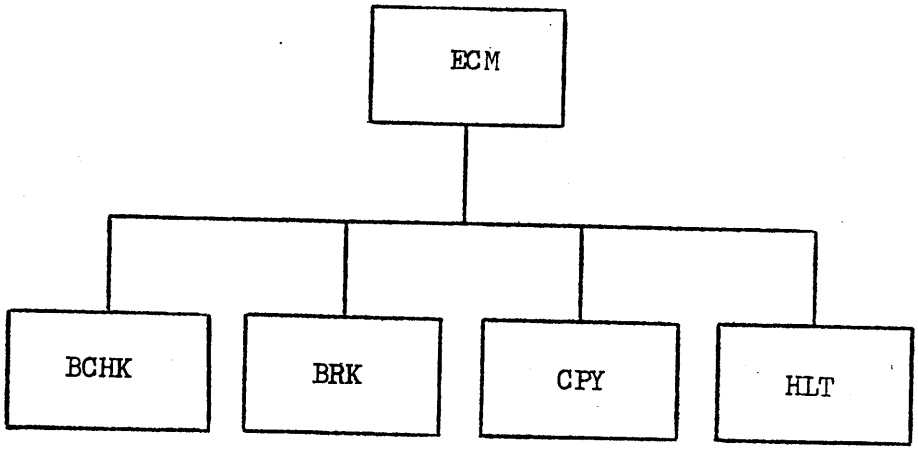


Fig. A7. Call tree for the command handler "ECM".

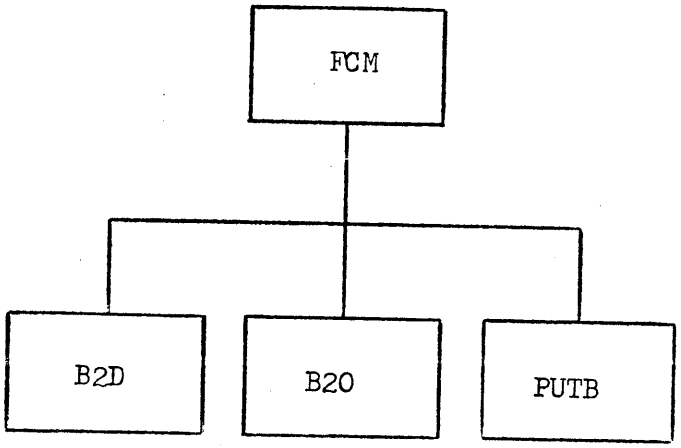


Fig. A8. Call tree for the command handler "FCM".

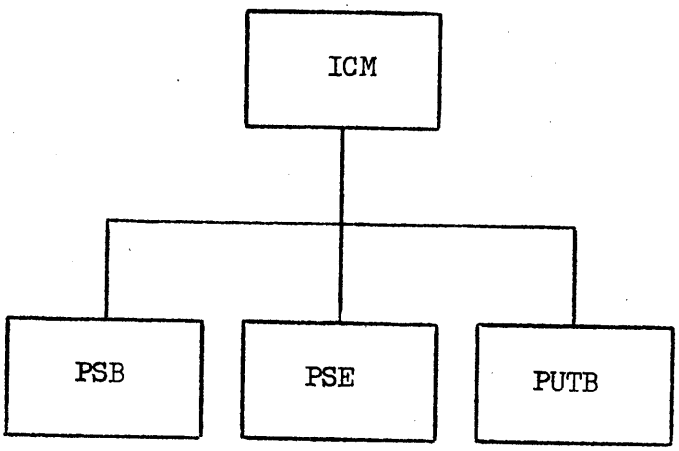


Fig. A9. Call tree for the command handler "ICM".

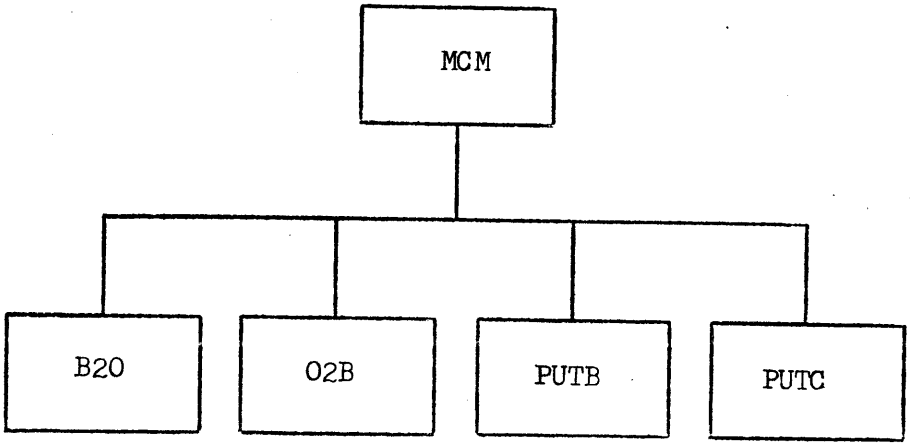


Fig. A10. Call tree for the command handler "MCM".

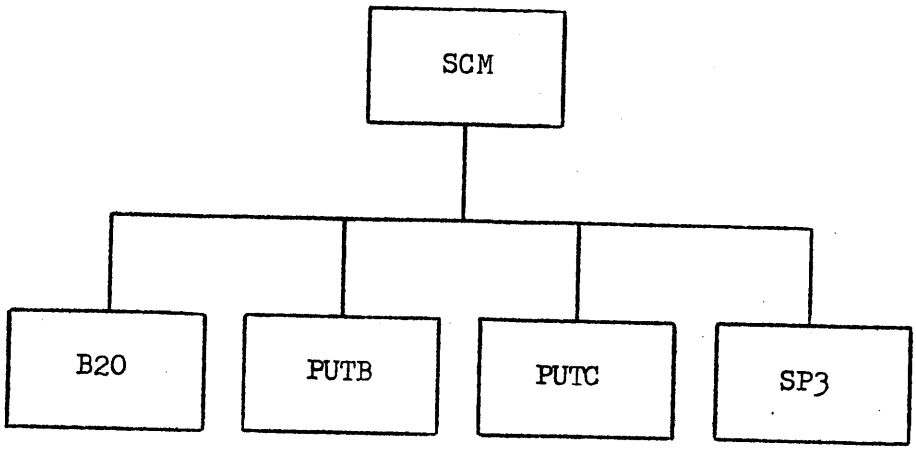


Fig. A11. Call tree for the command handler "SCM".

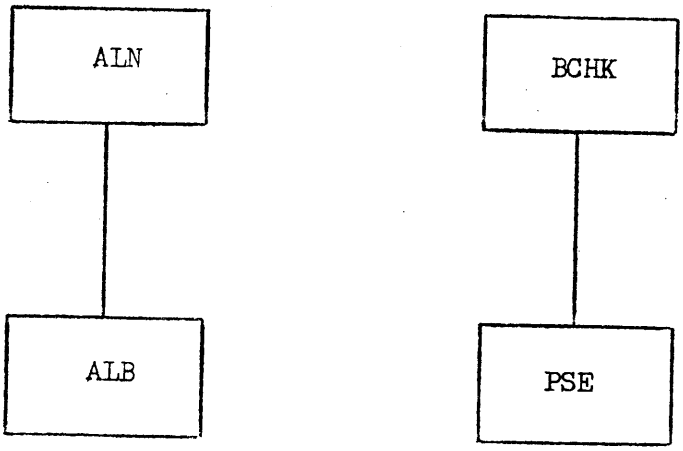


Fig. A12. Call trees for the utilities "ALN" and "BCHK".

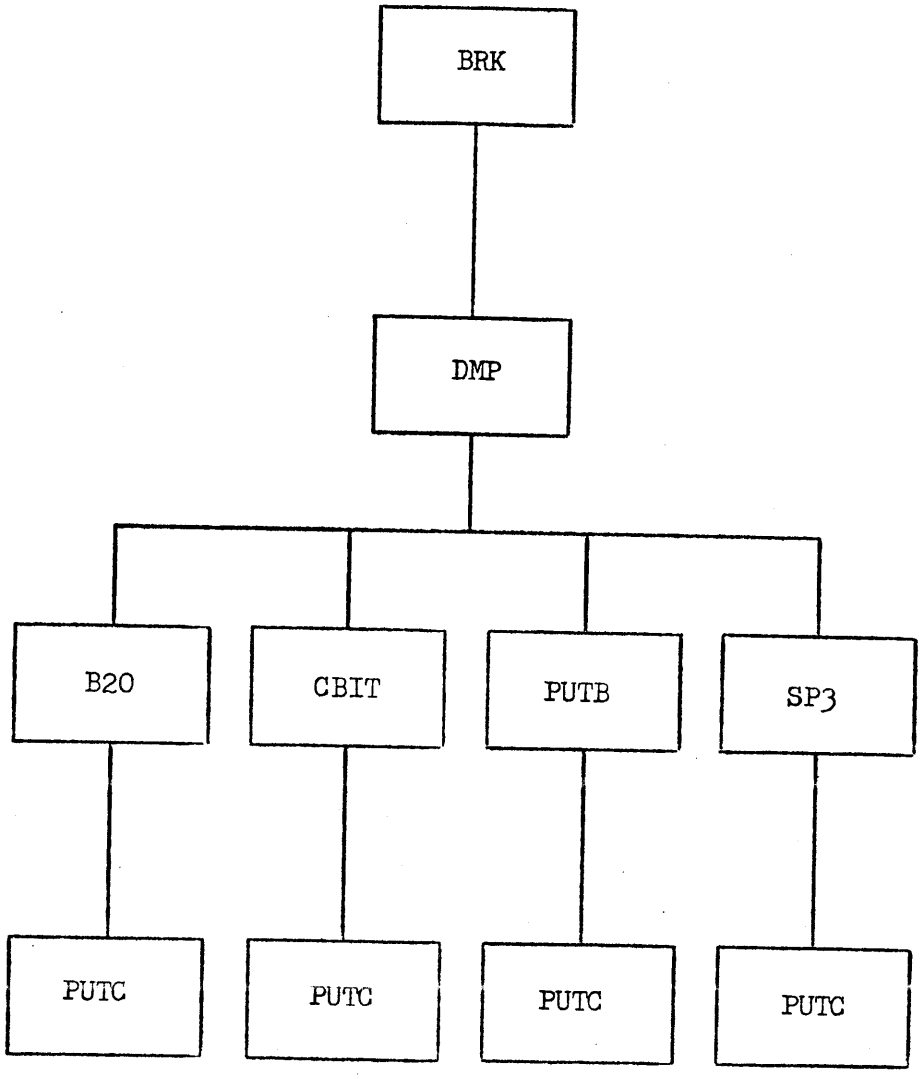


Fig. A13. Call tree for the utility "BRK".

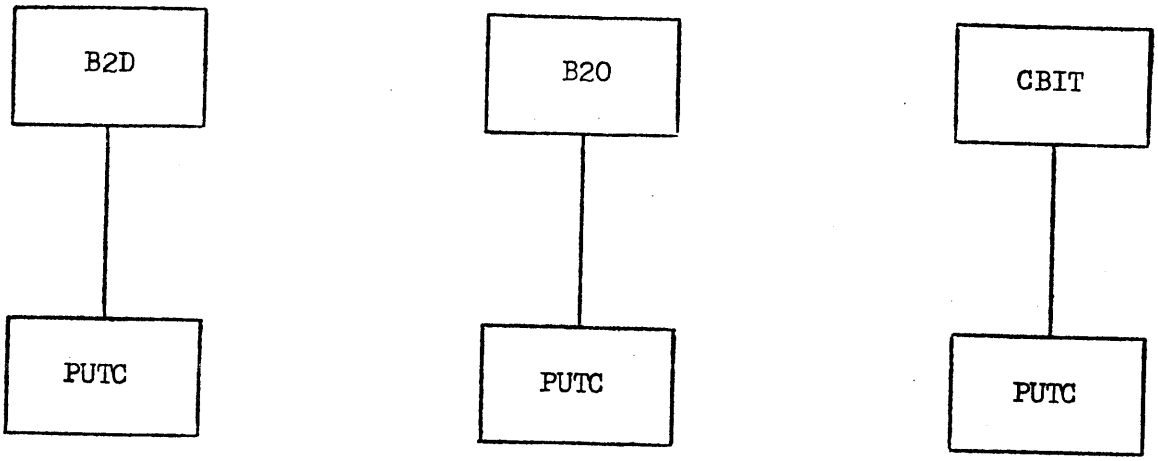


Fig. A14. Call trees for the utilities "B2D", "B2O" and "CBIT".

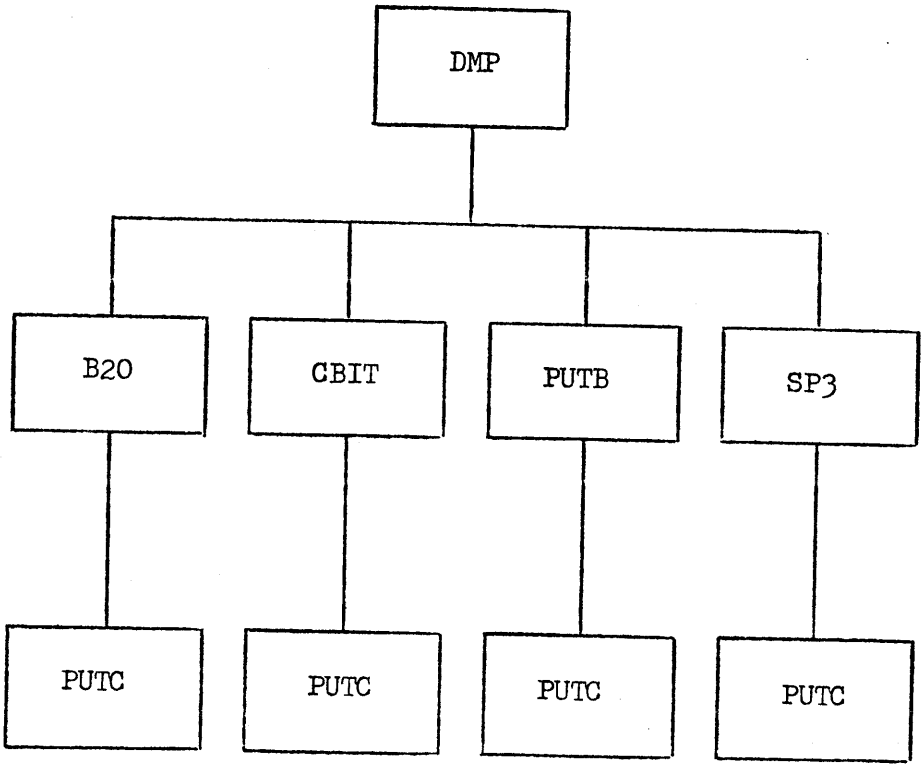


Fig. A15. Call tree for the utility "DMP".

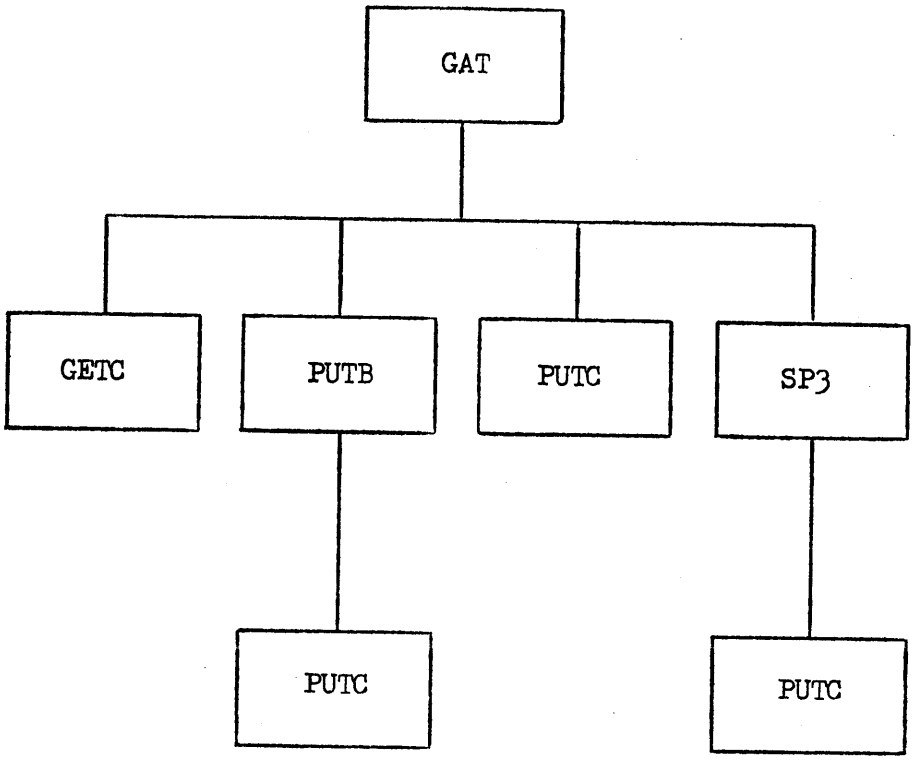


Fig. A16. Call tree for the utility "GAT".

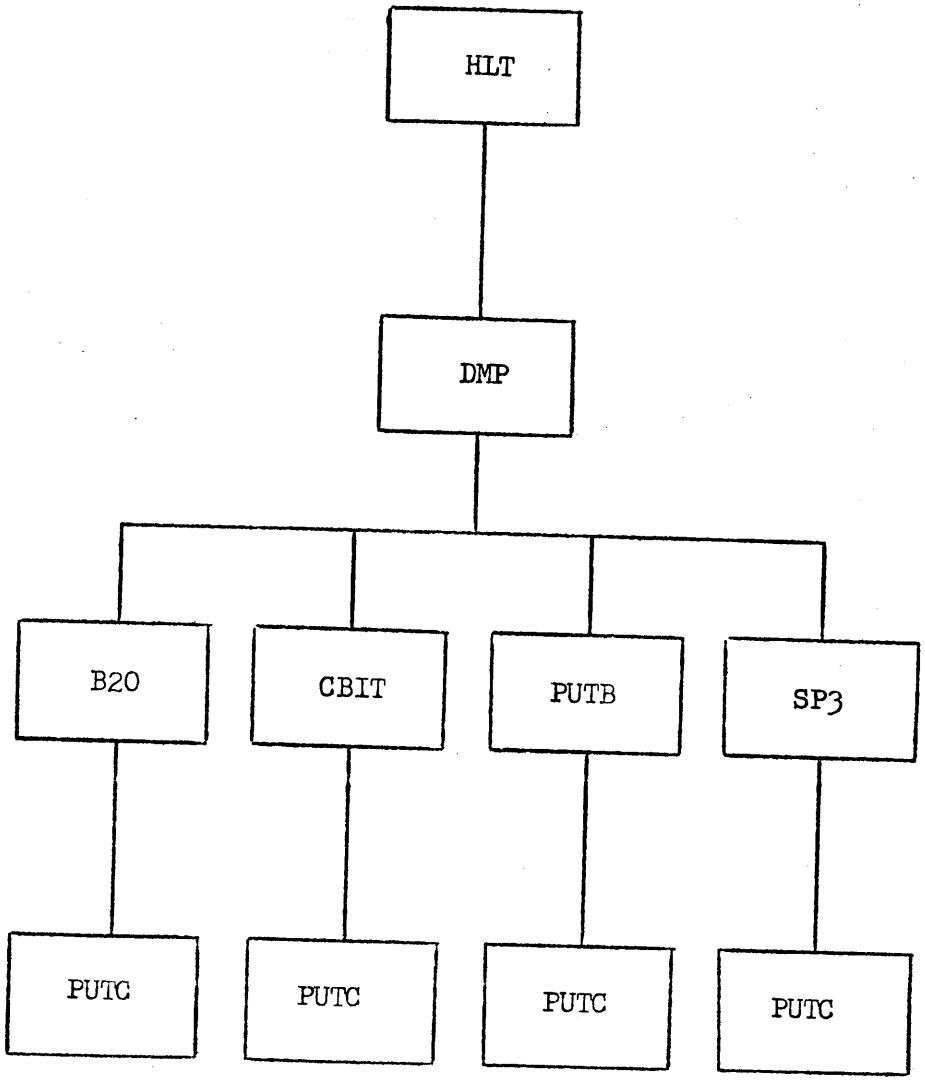


Fig. A17. Call tree for the utility "HLT".

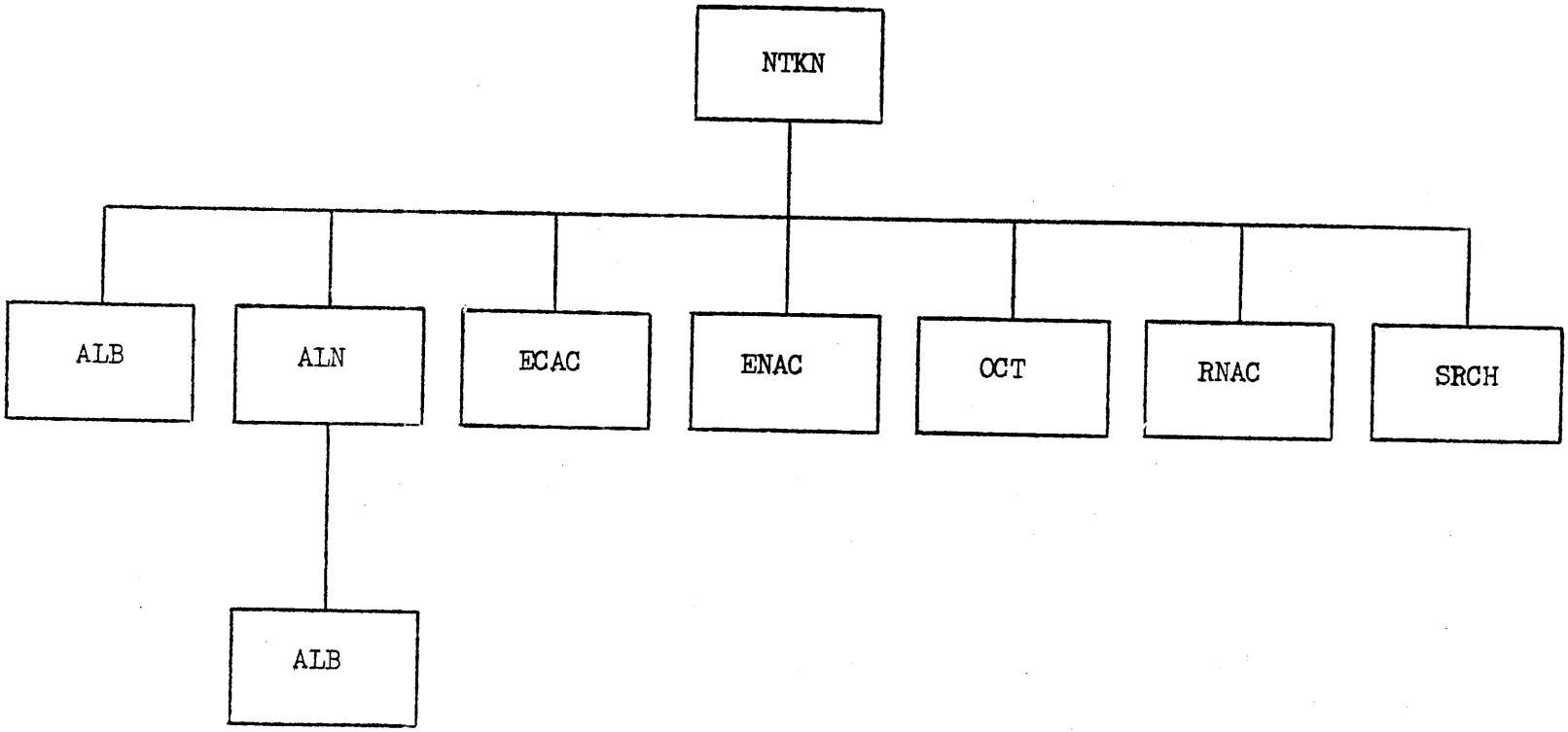


Fig. A18. Call tree for the utility "NTKN".

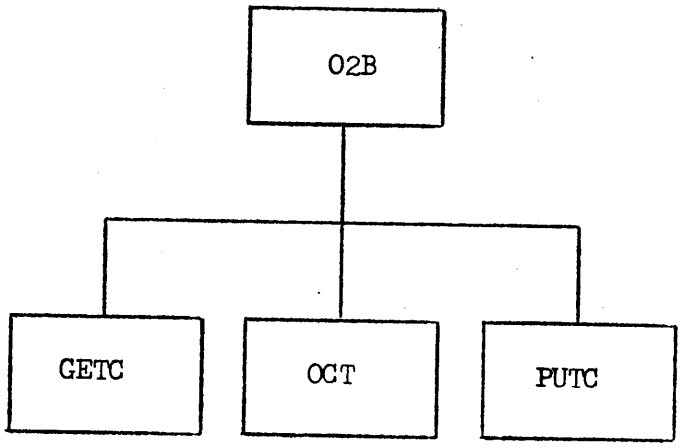


Fig. A19. Call tree for the utility "02B".

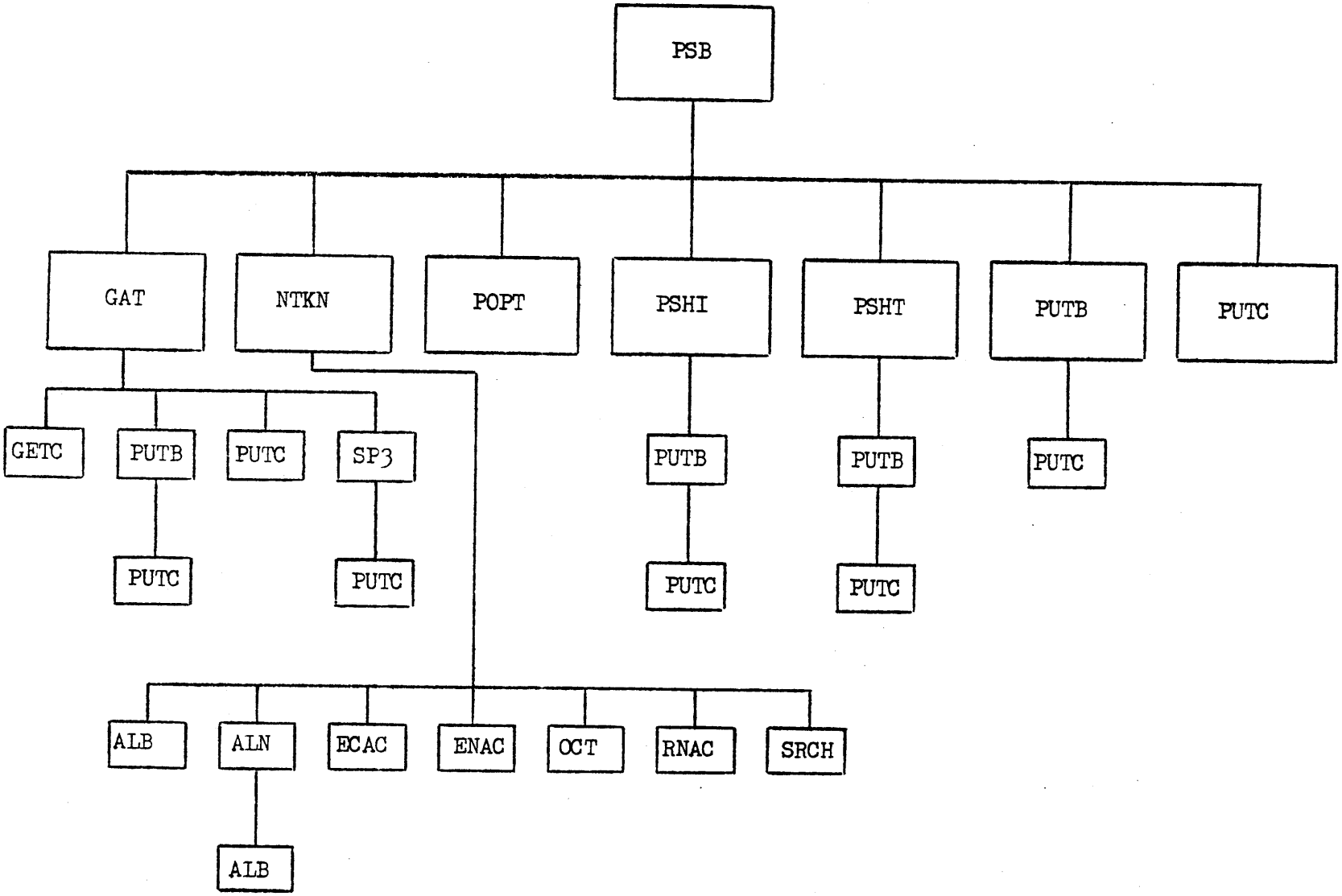


Fig. A20. Call tree for the utility "PSB".

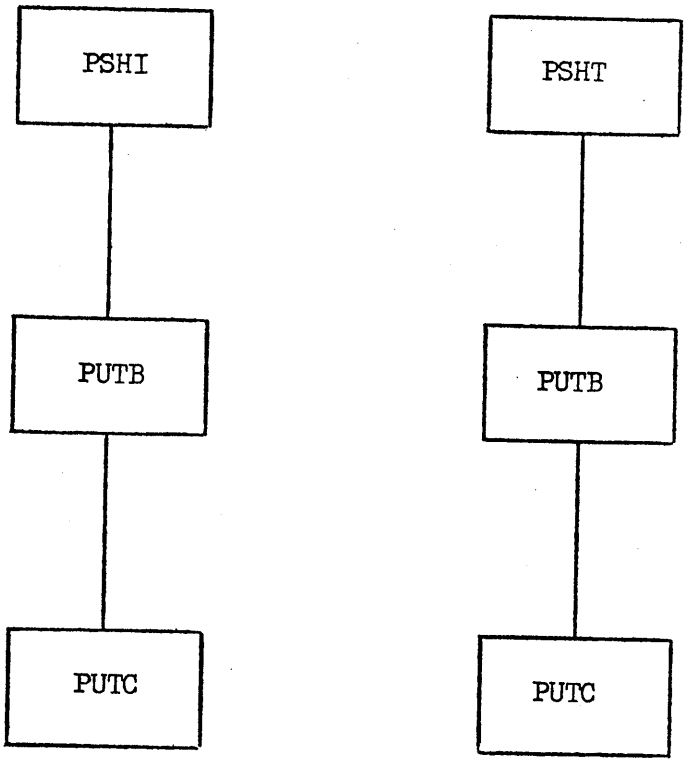


Fig. A21. Call trees for the utilities "PSHI" and "PSHT".

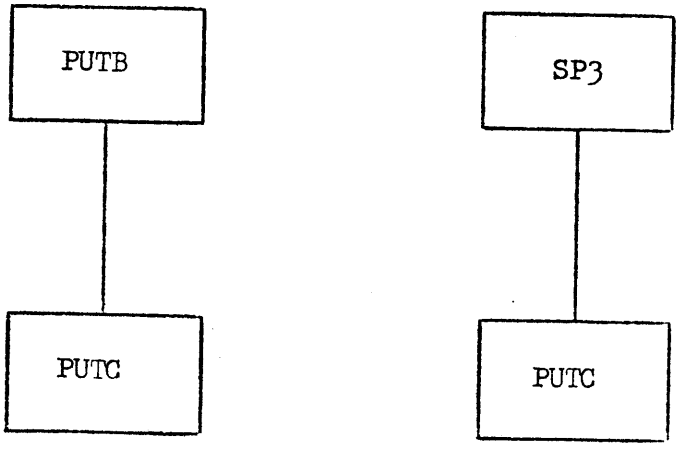


Fig. A22. Call trees for the utilities "PUTB" and "SP3".

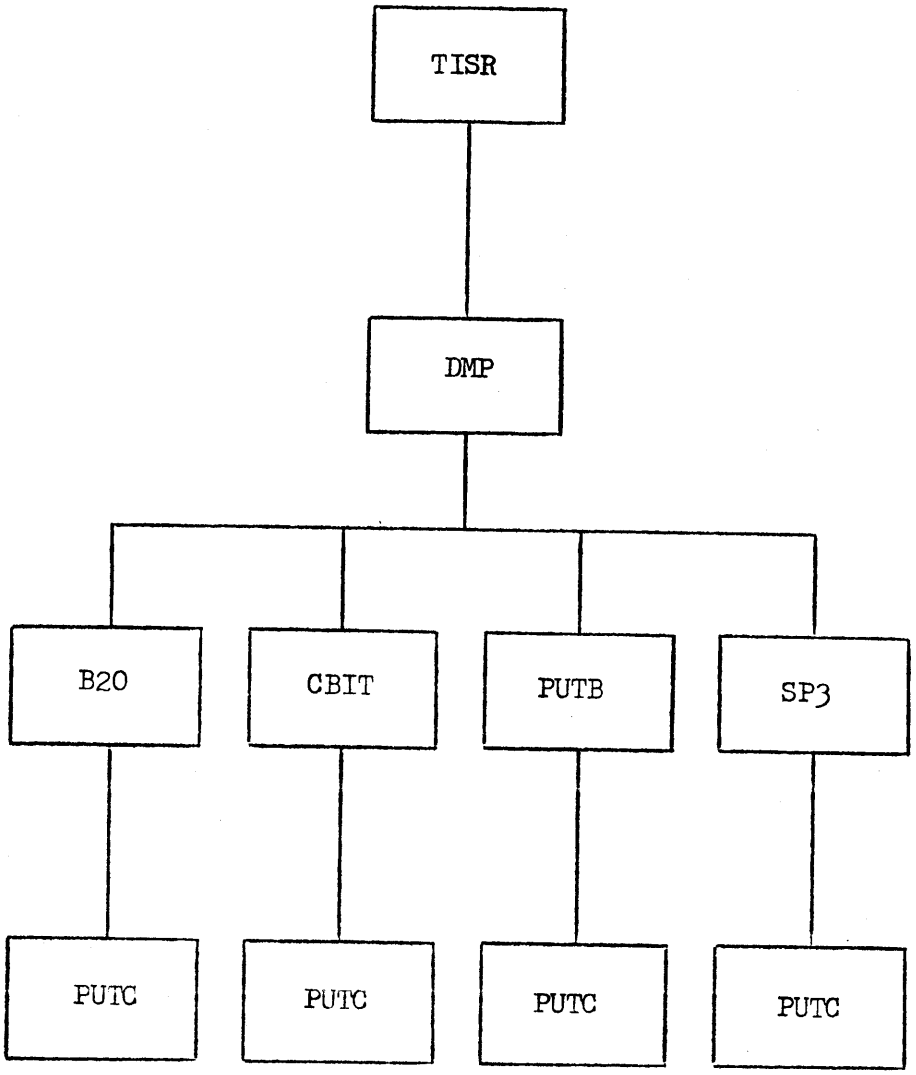


Fig. A23. Call tree for the utility "TISR".

APPENDIX F: ALADDIN STACK FORMATS

The figures in this appendix show the internal stack formats used in the NOVA implementation of ALADDIN.

Entry Code	Entry Value

(2-word Polish stack entry)

Entry type	Entry code	Entry value
Fence	-1	-----
Unused	0	-----
Operator	1	Operator's ID
Symbolic operand	2	Symbol table address
Literal operand	4	Literal's octal value

Fig. A24. Polish stack internal format.

Operator ID	Operator Priority

(2-word temporary operator stack entry)

Operator	ID	Priority
"$"	-1	0
"+"	0	4
"-"	1	4
"="	2	3
"<>"	3	3
"<"	4	3
">"	5	3
"<="	6	3
">="	7	3
"&"	8	2
":"	9	1
"("	10	0
")"	11	0

Fig. A25. Temporary operator stack internal format.

DATAPOINT CORPORATION

January 7, 1980

Professor Richard E. Fairley
Computer Science Department
Colorado State University
Fort Collins, Colorado 80521

Dear Professor Fairley:

May I have a copy of the listings and documentation on your ALADDIN system, as offered in your paper on that system in IEEE Transactions on Software Engineering for July 1979. Thank you.

Yours sincerely,

Mark Halpern

Mark Halpern

MH:jl

WESTERN DEVELOPMENT CENTER 2126 SIXTH ST. BERKELEY, CALIFORNIA 94710 (415) 848-7100

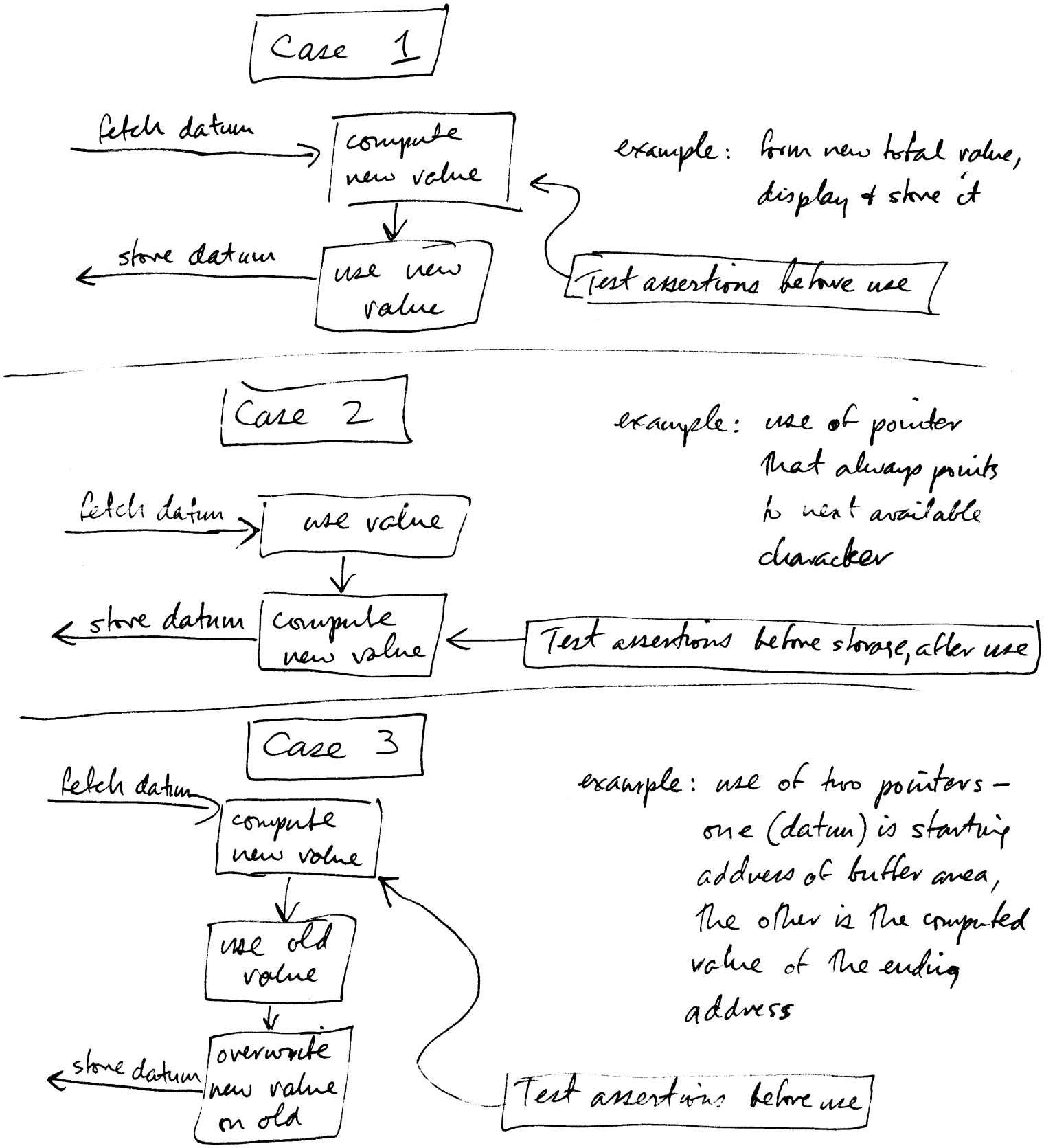
SBF doesn't work for bugs:

① That consist of passing syntactically correct but semantically wrong values (e.g. SPR #2178)

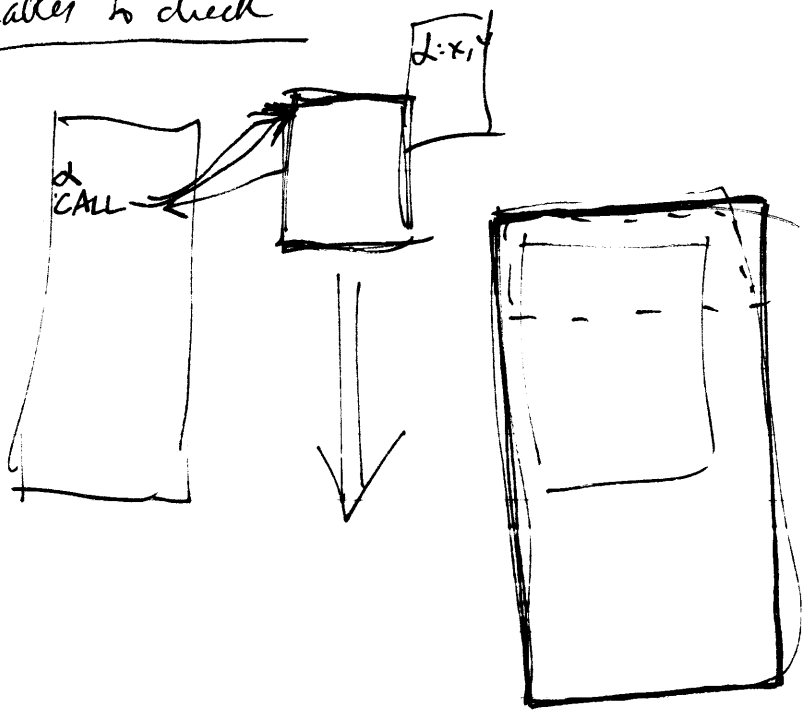
② That consist of using assembly-control directives in a syntactically correct but semantically wrong way (e.g., our fix to the bug reported in SPR #2178)

All above ↑ is wrong! Cases ① and ② potentially generate deviations in numeric values just like any other bug. The task is to track their consequences until such deviations occur, then establish: (a) what state program is in, (b) whether debugging person can readily backtrack to the bug.

True (?): Code must load properly before SBF can help!


Case 1
fetch datum
compute new value
store datum
use new value
Test assertions before use
example: form new total value, display & store it
Case 2
example: use of pointer that always points to next available character
fetch datum
use value
store datum
compute new value
Test assertions before storage, after use
Case 3
fetch datum
compute new value
use old value
overwrite new value on old
store datum
Test assertions before use
example: use of two pointers – one (datum) is starting address of buffer area, the other is the computed value of the ending address

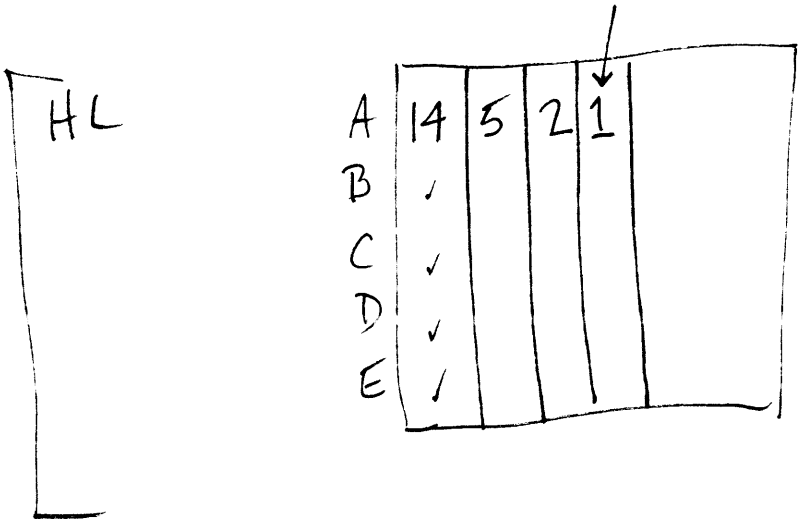
variables to check



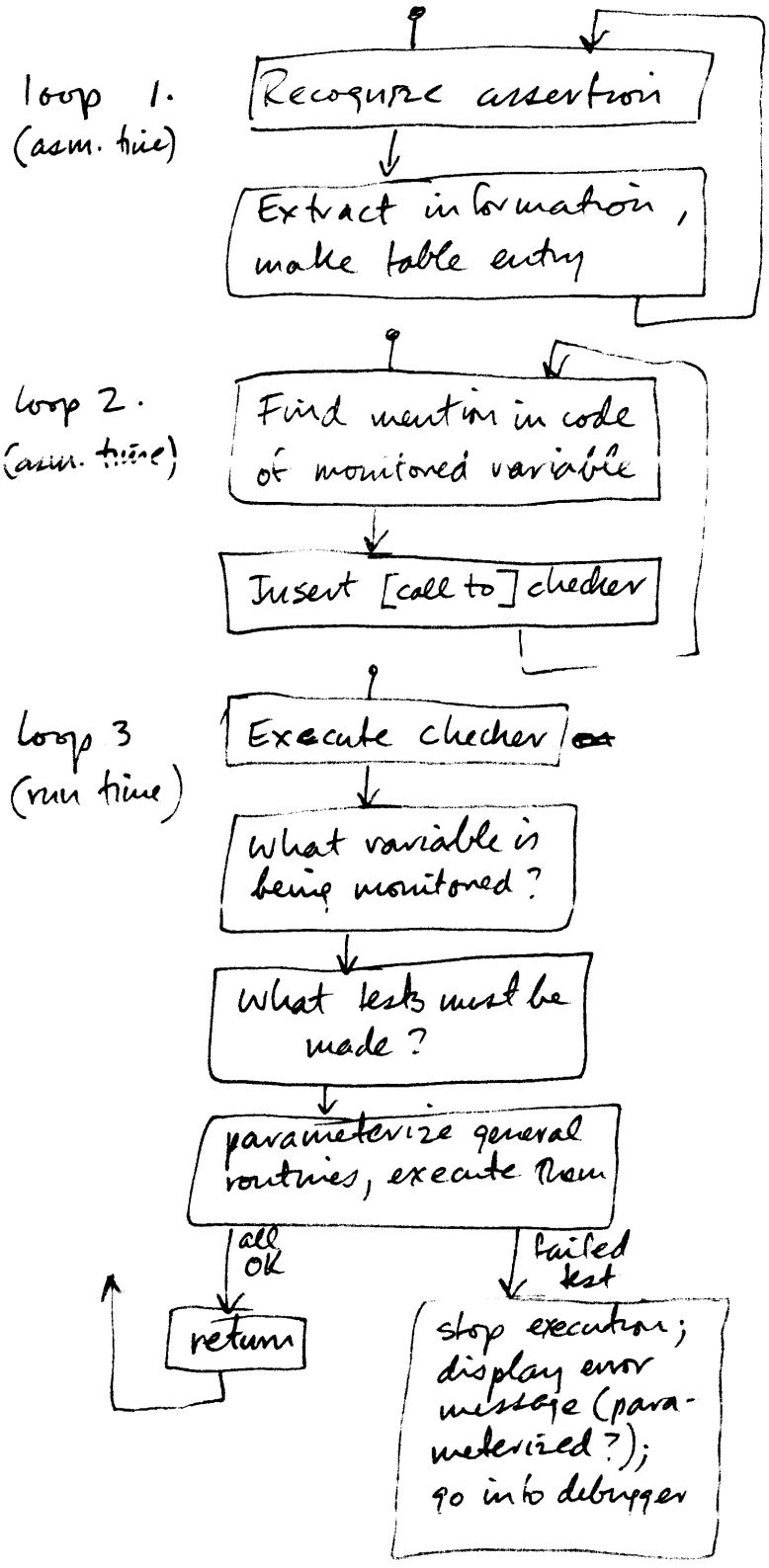
ALPHA = ALPHA + 1
CALL CHECKER, ALPHA

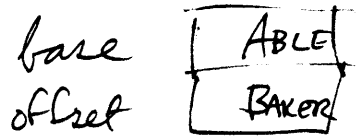
1	8	64	512	~~40~~ 96	32768
6	4	2	0	0	1

32768
128
32
6



Minimal "SNAP4" Specs.






```
LA      ABLE
LB      BAKER
ADBA                (ABLE + BAKER)
DE      TABLE
```

```
HL      BASE
LA      OFFSET
INCP    HL, A
LAM
```

BAS

"SNAP4" produces: (a) list of variables to be monitored
(b) assertions (ie, compiled routines in library) to be checked for each such variable

---

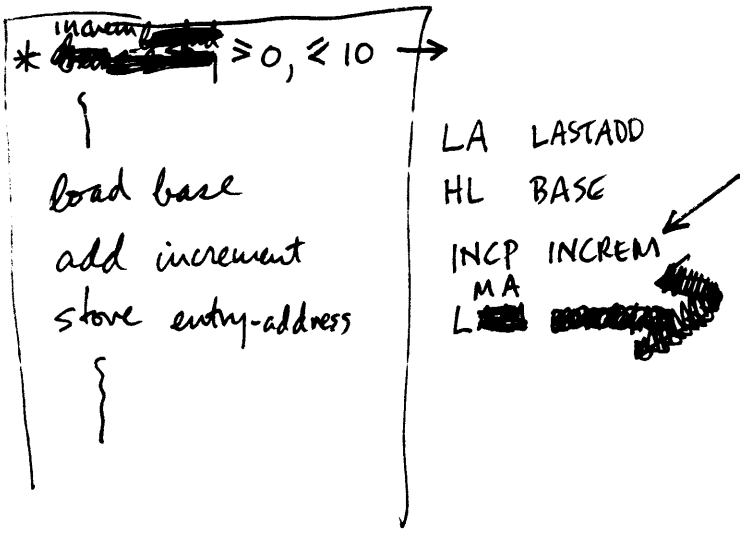
SNAP4 accepts assertions embedded (at beginning?) in source code, makes list of variables constituting the subjects of these assertions, finds all instances of storage of values into those variables (by explicit name only, or by base+offset as well?), inserts just before each such instance a CALL to a routine that

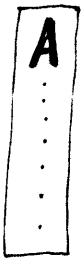
① [redacted] ② ~~Make assertions in the source code~~

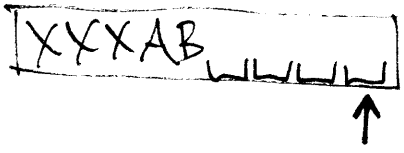
SNAP4

assembly-time

① scans source code for possible problems; e.g., unbalanced IFSET-XIF's summarizes all error reports for an assembly







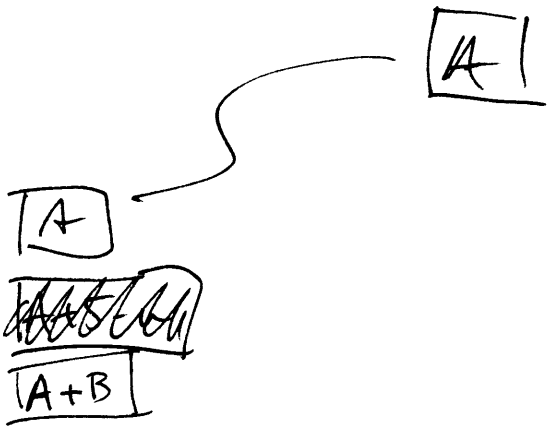
X = A, B, C, D

object-time

① accepts assertions in source code that generate tests (or calls) in the object code:

1. Recognize, decompose assertion; make appropriate table entries [allow forward refs?]
2. Inserts code at point(s) where variable is ~~fe~~stored. [variable explicitly named?]
3. Library of assertion validity-testing routines must be available at run time (1st priority: VALUE-RANGE CHECK)

	I	II
HL	CALL HL	
LMA	~~CALL~~ LMA	
SUC	CALL SUC	



① value-/range limits
② change only by ⟨step-size⟩

ASSERTION-TYPE	REQUIRES
① VALUE $x \leq n \leq y$	storage of x, y
② STEP-SIZE = DELTA	" " DELTA $\{\pm\}$? cross-check with MONOTONIC?
③ MONOTONICITY	" " last value
④ TARGET OF JUMP IS INSTRUCTION	